

# Superskalare Architekturen: Vom CDC 6600 zu den modernen Mikroprozessoren

Technische Universität München SS2019  
Lehrstuhl für Rechnerarchitektur und Parallele Systeme  
Seminar: Geschichte der Rechnerarchitektur

Cecilio C. Tamarit  
Universidad Politécnica de Valencia  
Valencia, Spanien  
cetaca@inf.upv.es

Dirk Münzenmaier  
Technische Universität München  
München, Deutschland  
dirk.muenzenmaier@tum.de

## ABSTRACT

Superskalare Prozessoren führen mehrere Befehle pro Zyklus aus, um die Parallelität auf Befehlsebene (Instruction-Level Parallelism oder ILP) auszunutzen und damit einen höheren Durchsatz (IPC) und eine höhere Leistung zu erzielen. Obwohl die ersten Implementierungen in den sechziger Jahren vorgenommen wurden, kann die derzeitige Relevanz der Superskalarität nicht geleugnet werden, da heutzutage fast jeder Prozessor diese Technik einsetzt, es sei denn, Leistungs- oder Kostenbeschränkungen sind wirklich anspruchsvoll. Dieses Paper wurde mit der Absicht verfasst, diesen wichtigen Beitrag zur Rechnerarchitektur in ihren historischen und technologischen Kontext zu stellen.

## CCS Concepts

•Computer systems organization → Superscalar architectures;

## Keywords

Rechnerarchitektur; superskalar; superskalare Architekturen; Parallelität auf Befehlsebene; ILP; Instruction-Level Parallelism; pipelining; Cray; Control Data Corporation; CDC6600; IBM; IBM System/360 Model 91

## 1. EINFÜHRUNG

**Superskalare Prozessoren** führen mehrere Befehle pro Zyklus aus, um die Parallelität auf Befehlsebene (Instruction-Level Parallelism oder ILP) auszunutzen und damit einen höheren Durchsatz (Instruktionen pro Zyklus oder auf Englisch IPC) und eine höhere Leistung zu erzielen.

**Parallelität auf Befehlsebene** und insbesondere superskalare Prozessoren sind relevante Themen in der Geschichte der Rechnerarchitektur. Diese Techniken tauchten in den sechziger Jahren auf, fanden aber erst in den achtziger Jahren ihren Weg zur kommerziellen und breiten Akzeptanz [31]. Seitdem hat ILP eine zentrale Rolle bei der Entwicklung neuer Prozessoren gespielt, und sogar heutzutage implementieren fast alle CPUs auf dem Markt ILP oder Out-of-Order-Execution von Instruktionen, sofern keine strengen Leistungs- oder Kostenbeschränkungen gelten (z. B. im Bereich eingebetteter Systeme).

In dieser Seminararbeit werden von Rechner wie der CDC 6600 und ihre Inspirationen in den sechziger Jahren bis hin zum moderneren Intel Pentium erklärt, um diese heutzutage auch wichtige Entwicklung in ihren historischen Kontext zu stellen. Um zu verstehen, wie superskalare Prozessoren funktionieren, müssen aber zunächst andere allgemeinere Konzepte auf dem Gebiet der Rechnerarchitektur erläutert werden.

### 1.1 Parallelität auf Befehlsebene (ILP)

In Prozessoren und Compilern, die für Parallelität auf Befehlsebene konzipiert sind, verarbeitet der Computer normalen, sequentiellen Code und zielt darauf ab, ihn schneller auszuführen, indem er transparent mehr als eine Operation während desselben Taktzyklus ausführt. Diese Operationen können Arithmetik, Speicherladevorgänge oder sogar Verzweigungen sein.

Superskalarität ist eine Art ILP. Der Hauptunterschied zwischen einem Prozessor, der Superskalarität nicht verwendet, und einem, der es ausnutzt, besteht darin, dass dieser mehr als eine der gleichen Ausführungseinheiten aufweist. Um beispielsweise vier Additionen gleichzeitig verarbeiten zu können, müssen vier Addierer gleichzeitig verfügbar sein.

Parallelität auf Befehlsebene kann in drei verschiedene Hauptkategorien unterteilt werden [31]:

- **Sequentielle Architekturen**  
Die Hardware verwaltet die Parallelität, während der Code gleich bleibt. (Superskalare Prozessoren)
- **Abhängigkeitsarchitekturen**  
Operationen mit Abhängigkeiten werden im Code explizit angegeben. (Dataflow-Prozessoren)
- **Unabhängigkeitsarchitekturen**  
Das Gegenteil der vorherigen. Unabhängige Operationen sind im Code explizit angegeben. (VLIW-Prozessoren)

In dieser Seminararbeit werden wir uns mit dem ersten beschäftigen. Zunächst wird das Konzept erläutert und im Anschluss daran sind Beispiele von Computern gegeben, die diese Techniken implementiert haben, von seiner frühen Geschichte bis zur Gegenwart und wie es Computer von heute beeinflusste.

Befehl / Zyklus	1	2	3	4	5	6	7	8
i1 add r1, r2, r3	IF	ID	EX	M	WB			
i2 add r4, r5, r6		IF	ID	EX	M	WB		
i3 add r7, r8, r9			IF	ID	EX	M	WB	

Table 1: Simple RISC *instruction pipeline*. In diesem Beispiel ist es in fünf Phasen unterteilt: Abrufen, Dekodieren, Ausführen, Speichern und Write-back.

Befehl / Zyklus	1	2	3	4	5	6	7	8
i1 add r1, r2, r3	IF	ID	EX	M	WB			
i2 add r4, r5, r6	IF	ID	EX	M	WB			
i3 add r7, r8, r9		IF	ID	EX	M	WB		

Table 2: *Two-way-superskalare instruction pipeline* mit (wenigstens) zwei Additionseinheiten

## 1.2 Pipelining und superskalarität

Superskalare Prozessoren sind, wie bereits mehrfach ausgeführt, eine Form der Parallelität auf Befehlsebene, die insbesondere zur Kategorie der sequentiellen Architekturen gehört. Um das Konzept richtig zu verstehen, ist es zuerst wichtig zu wissen, was „instruction pipelining“ ist.

Das **Pipelining**, das erstmals in der IBM 7030 („Stretch“) eingeführt wurde, ermöglicht es uns, mehrere Befehle gleichzeitig auszuführen. Wir tun dies, indem wir Prozessoren in verschiedene Phasen unterteilen, die unterschiedliche Aufgaben (oder Teile davon) machen. Beispielsweise befindet sich in der Tabelle 1 der Befehl 1 in der Ausführungsphase, während der Befehl 2 decodiert und der Befehl 3 „gefetched“ wird. Eine Pipeline wie diese ist eine andere Form der Parallelität auf Befehlsebene, aber superskalare Prozessoren können sein und sind in der Regel auch *pipelined*.

Superskalare Prozessoren haben den Vorteil, dass sie nicht nur eine einzelne Funktionseinheit ausnutzen und unterteilen, sondern auch mehr davon haben und auf diese Weise zwei oder mehr Befehle gleichzeitig in derselben Phase sein können. [41] [36] Beispielsweise könnte die vorherige Programmablaufverfolgung wie in der Tabelle 2 aussehen, wenn sie auf einem sogenannten *two-way-superskalaren* Prozessor ausgeführt wird. Die Anzahl der Befehle, die gleichzeitig abgerufen und ausgegeben werden können, hängt von der Anzahl der ways ab, die der superskalare Prozessor hat, auch als „Grad“ bezeichnet.

In diesem Fall können zwei Additionen gleichzeitig durchgeführt werden, da es mindestens zwei Addierer und zwei oder mehr andere Funktionseinheiten gibt. Auf diese Weise könnte das Programm in sechs Zyklen anstelle von sieben fertig sein. Oder 5, wenn es three-way-superskalar wäre und drei Addierern usw. hätte.

Vor superskalare Prozessoren war das Ziel immer, die Zyklen pro Befehl so zu reduzieren, dass es ungefähr nur 1 Zyklus war. Dieser Fokus wurde später durch das Erreichen von mehr als ein Befehl pro Zyklus ersetzt. [31] [17] Je breiter die Pipeline und auch so der Anzahl der Funktionseinheiten wächst, desto höher den IPC-Wert sein kann, da mehr Befehle gleichzeitig abgerufen, ausgegeben und ausgeführt

werden können.

Wie bei der Erläuterung der sequentiellen Architekturen hervorgehoben wurde, müssen keine Änderungen am Code vorgenommen werden, um diese Form der Parallelität auszunutzen. Es gibt aber eine wichtige Einschränkung: **Abhängigkeiten**. In sequentiellen Programmen sollen Befehle nacheinander ausgeführt werden, und wenn i2 vom Ergebnis von i1 abhängt, kann es nicht gleichzeitig ausgeführt werden. Das muss von der Hardware überprüft und gelöst werden. (Beachten Sie, dass es keine Abhängigkeiten zwischen den Anweisungen in den oberen Beispielen gibt.) In solchen Fällen müssten Stall-Zyklen eingefügt werden, bis das erforderliche Ergebnis verfügbar ist. Eine andere Möglichkeit, diese Abhängigkeiten zu lösen besteht darin, während der Kompilierungszeit die vorhergehende Befehle zu überprüfen und neu zu ordnen und, falls die Hardware nicht mit ihnen umgehen kann, nop-Befehle einzufügen.

## 1.3 Die drei Regeln des ILP

Laut Rau und Fisher (1993) [31] muss jeder Prozessor, der von ILP profitieren will, die folgenden drei Regeln einhalten:

- Die Abhängigkeiten zwischen Operationen müssen festgelegt werden.
- Die Operationen, die unabhängig von einer noch nicht abgeschlossenen Operation sind, müssen festgelegt werden.
- Diese unabhängigen Operationen müssen so geplant werden, dass sie zu einem bestimmten Zeitpunkt auf einer bestimmten Funktionseinheit ausgeführt werden, und ihnen muss ein Register zugewiesen werden, in dem das Ergebnis gespeichert werden kann.

Befolgen aber superskalare Prozessoren diese Regeln? Bei superskalaren Prozessoren werden die Regeln 1, 2 und 3 von der Hardware ausgeführt. In Abhängigkeitsarchitekturen erfüllt der Compiler Regel 1 und die Hardware übernimmt 2 und 3. In Unabhängigkeitsarchitekturen erledigt der Compiler schließlich alles und die Hardware muss den Code nur so ausführen, wie er ist, ohne etwas anderes zu überprüfen.

## 2. GESCHICHTE DER SUPERSKALARE ARCHITEKTUREN

ILP-ähnliche Techniken wurden seit den 40er Jahren angedeutet. Tatsächlich enthielt Alan Turings Pilot-ACE-Entwurf von 1946 einen horizontalen Microcode: Anweisungen, die zwei unterschiedliche Aufgaben gleichzeitig ausführen konnten. Da RISC-Prozessoren viel später relevanter wurden, wurden diese Techniken weniger üblich [48]. Seit der Entwicklung des Transistorcomputers in den 60er Jahren ermöglichte die exponentielle Zunahme der Anzahl der Logikgatter den Rechnerarchitekten, Parallelität auf Befehlsebene in ihre Entwürfe aufzunehmen. Die erste Implementierung, die man als einen superskalaren Computer betrachten kann, war der CDC 6600. Zusammen mit dem IBM System/360 Model 91 waren diese beiden Computer Vorreiter und ebneten den Weg für alle späteren.

### 2.1 Frühe Geschichte

#### 2.1.1 Entwicklung und Beiträge des CDC 6600

Der **CDC 6600** kann als Übergang von herkömmlichen Scalar-Computern zu superskalaren Computern betrachtet werden. Es wurde von Seymour Crays Team bei Control Data Corporation in den 60er Jahren entworfen und hatte zum Ziel, das „Supercomputer-Rennen“ gegen IBM et al. zu gewinnen, das begann das damals zu geschehen begonnen ist. Das Design war nicht so hochgesteckt als bei den folgenden Rechnern: eine Issue-Rate von 1 IPC wurde beibehalten. (Denken Sie daran, dass dies vor dem oben erwähnten Mentalitätswechsel von CPI zu IPC geschah.) Aus diesem Grund war es ein Pionier, ILP mit der Hinzufügung von mehr als einer der gleichen Funktionseinheiten zu haben, aber immer noch nur ein Befehl pro Zyklus abgerufen („fetched“) und ausgegeben („issued“).

#### *Inspiration und erstes Entwurf*

Das Entwurf dieser Maschine begann 1960. Es war eine Zeit großer technologischer Fortschritte, hauptsächlich aufgrund des Weltraumrennens zwischen den USA und der UdSSR. Die enormen Ausgaben für Forschung und Entwicklung führten zur Entwicklung und Einsatz von Satelliten für Wetter und Kommunikation und natürlich auch von Raumfahrtstechnologien, die Operationen wie die Apollo 11-Mission ermöglichten, mit denen die Menschen im selben Jahrzehnt

zum ersten Mal zum Mond gebracht wurden. In Bezug auf Computer wurden integrierte Schaltkreise nur drei Jahre vor der Gründung der **Control Data Corporation** in Minneapolis erfunden [44] [43]. Bis 1960 wurden die ersten großflächigen Computer mit Transistoren herausgebracht, nämlich der IBM 7090 und später der CDC 1604, beide für den Absatzmarkt der wissenschaftlichen Computer. Letzteres Entwurf wurde von **Seymour Cray** gerichtet, und beide sind in der Figur 1 zu sehen.

**James E. Thornton** war zu dieser Zeit ein weiterer Mitarbeiter, der unter Cray arbeitete und andere Ingenieure des Projektes CDC 1604 überwachte [44]. Er besuchte ein Seminar an der UCLA, das sich mit den neuesten Trends bei Supercomputern befasste. Dort lernte er das Innenleben des noch zu veröffentlichenden IBM-Codenames „Stretch“, der ILLIAC II usw. kennen. Laut Thornton war die technische Literatur zu dieser Zeit begrenzt, und dies war der beste Weg, mehr Einblick in den neuesten Entwicklungen zu gewinnen. Die Technologie hat sich damals, wie bereits erwähnt, sehr schnell immer weiterentwickelt, und deswegen war es sinnvoll. Nach seiner Rückkehr lud ihn Cray in ein neues Team ein, dessen Ziel war den CDC 6600 zu entwickeln, den CDC-Computer der nächsten Generation.

Manchmal muss die Kompatibilität zum Wohle der Leistung geopfert werden [12]. Es war bereits ein kompatibles Upgrade für den 1604 in Arbeit, die 3000er-Serie, aber der 6600 musste komplett anders sein. Aus diesem Grund bildeten sie ein kleines, separates Team und zogen in ein anderes Gebäude, um sich vom Rest des Unternehmens zu isolieren. Mit den Insiderinformationen, die Thornton während des Seminars erhalten hatte, **war es ihr Ziel, sich mit IBM auf dem Markt für wissenschaftliche Computer zu messen**. Um dies zu erreichen, würde die Verwendung schnellerer Transistoren nicht ausreichen: Sowohl Stretch als auch LARC würden **neue architektonische Fortschritte** und in gewisser Maße Parallelität (meist in Form von Pipelining) implementieren, daher mussten sie dies auch tun [44]. In der Zwischenzeit waren sowohl der 1604 als auch seine Upgrades, auch wenn sie nicht so innovativ waren, sehr erfolgreich und unterstützten die Entwicklung des 6600-Projekts in den kommenden Jahren, da dieser Prozess aufgrund vieler Hindernisse länger als erwartet dauern würde. In der Tat wurde das Projekt nach fast einjähriger Arbeit aufgrund von Designfehlern und anderer Schwierigkeiten abgebrochen, nachdem sie sich bereits auf die zu verwendende Technologie und



Figure 1: Links, CDC 1604. Rechts, Seymour Cray. (Bilder: ithistory.org bzw. Star Tribune)

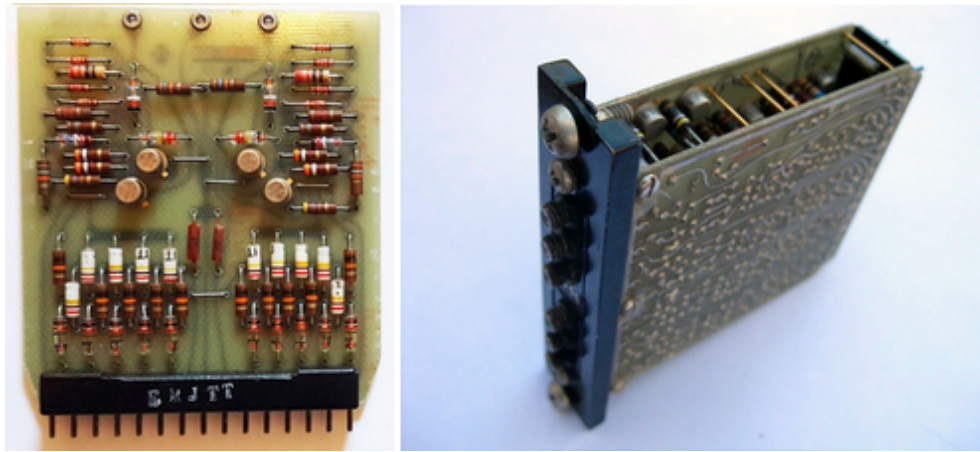


Figure 2: Links, sogenannte „Bausteine“, die CDC vorher benutzte. Rechts, „Cordwood-Paket“, dichter und zweiseitig. (Bilder: VAXBARN bzw. David Forbes, Creative Commons)

Struktur geeignet hatten und auch mehrere Tests durchgeführt hatten.

### *Fortschritt der Technologie bringt das Projekt wieder zum Leben*

Einige Wochen später begann Fairchild Semiconductor mit der Herstellung von **Transistoren aus Silizium**. Es ist mittlerweile zum Standardmaterial geworden, aber zu dieser Zeit basierten diese Komponenten normalerweise auf Germanium. Es wurde festgestellt, dass die Verkabelung zwischen den Logikgattern zu einem „Bottleneck“ geworden war. Silizium würde es ihnen ermöglichen, die Kabellänge zu reduzieren, um die Kommunikation zu verbessern und die Packungsdichte zu erhöhen. So wurde das Projekt 1961 wiederbelebt [44]. Die Prozessoren früherer CDC-Modelle bestanden aus sogenannten „**Bausteine**“ (Figur 2), Leiterplatten, die aus verschiedenen Bauteilen bestanden, bei denen es sich jeweils um ein oder zwei Logikgatter handelte. Dieser modulare Ansatz ist sehr praktisch, wenn eine einzelne Komponente ausgetauscht oder die Maschine erweitert werden muss. Der 6600 hätte zu viele dieser Bausteine benötigt (zehn- bis zwanzigmal mehr als die Vorgängermodelle), und deswegen wurde ein neueres sogenanntes „**Cordwood-Paket**“ entworfen (Figur 2). Diese bestand aus zwei parallele Leiterplatten, die komplexer waren. So wurde eine 10-fach höhere Dichte erreicht. Mit dieser neuen Technologie auf Siliziumbasis und einem **verbesserten Kühlsystem** auf Basis von Freon-Gasröhren konnten sie ihre Leistungsziele erreichen [43][44].

Diese technologischen Fortschritte reichten mehr oder weniger, um der Leistung der auf dem Seminar diskutierten Maschinen zu erreichen, aber ihr Ziel war es, sie bei weitem zu übertreffen. Wie bereits erwähnt, mussten architektonische Änderungen vorgenommen werden. Eine der umstrittensten Ideen war, die I/O-Vorgänge von der CPU selbst zu trennen und sie an zehn Co-Prozessoren zu delegieren, die als Peripheral Processing Units bezeichnet werden. Cray selbst entwarf die „**barrel PPUs**“ [43], denen während der Ausführung eines Programms I/O-Operationen zugewiesen wurden, während die CPU die normale Ausführung fortsetzte oder zu einer anderen Task wechselte. Mit 10 PPUs konnten 10 I/O-Vorgänge gleichzeitig ausgeführt werden. Ei-

ne PPU könnte theoretisch auch Betriebssystemfunktionen ausführen, ohne das laufende Programm zu unterbrechen.

### *Entwurf der CPU*

In Bezug auf die CPU war Thornton für die Leitung des Designs verantwortlich, wobei der Schwerpunkt auch auf der Parallelität lag. Sein Befehlssatz war für seine Zeit einfach und orientierte sich am wissenschaftlichen Rechnen, das stark von Gleitkommaoperationen abhängt. **10 Funktionseinheiten** waren enthalten: (Figur 2) zwei Inkrementeinheiten für Adressen- und Ganzzahlberechnungen, zwei schnelle Gleitkomma-Multiplikationseinheiten, ein Addierer, ein Long-addierer, ein Shifter, eine Boolesche Einheit und eine Verzweigungseinheit. Alle diese Funktionen konnten parallel ausgeführt werden und **waren die Hauptquelle für ILP im CDC 6600**. Thornton gibt zu, dass sie möglicherweise zu ehrgeizig waren, da nur relativ wenige Programme ILP in vollem Umfang nutzen konnten. Als die Ingenieure mit der Technik vertraut wurden und fortschrittlichere FORTRAN-Compiler zur Verfügung standen, konnte die Software weiter optimiert werden, um diese Funktionen nutzen zu können [44]. Um jede Operation einer funktionalen Einheit zuzuordnen, entwickelten sie die „Anzeigetafel“ („**Scoreboard**“), mit der das Steuerungseinheit Register und Einheiten verfolgen und reservieren kann. Befehle wurden ausgeführt, sobald die benötigte Einheit frei und die für die Operation erforderlichen Register verfügbar waren [31]. Wie diese Einheiten miteinander verbunden waren kann man in der Figur 3 betrachten.

Zu den weiteren Neuerungen gehört ein „**exchange jump**“ Befehl, der vollständige Zustandsänderungen über Hardware für ein schnelles Kontextwechsel, und auch eine **Wortlänge von 60 Bit**. Letzteres war auch eine polemische Entscheidung. Cray und Thornton hatten mehr Erfahrung mit dem Oktalsystem und bevorzugten es auf diese Weise, argumentierten jedoch auch, dass es sich auch um eine Frage der Effizienz und der Einfachheit des Befehlssatzes handele [44].

### *Rezeption und Schlussfolgerungen*

Während die Entwicklung voranschritt, war das Team im Chippewa Falls Laboratory isoliert worden, dieses war für die Öffentlichkeit geschlossen. Der erste Kunde, der das Ge-

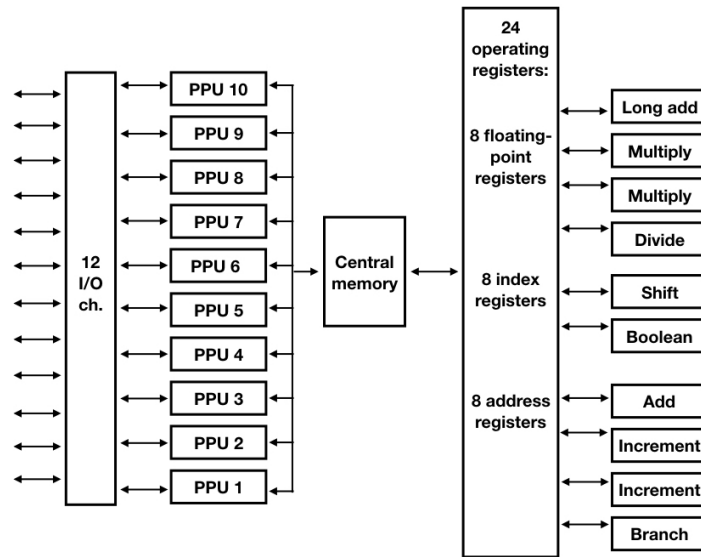


Figure 3: Simplifiziertes Blockdiagramm des CDC 6600

lände besichtigen durfte, das **Lawrence Livermore National Laboratory**, benötigte die leistungsstärkste Maschine auf dem Markt. Der CDC 6600 erfüllte ihre Bedürfnisse. Erst 1963 wurde ein vollständiger Prototyp gebaut und der Computer in einer Pressekonferenz offiziell veröffentlicht. Es wurde 1964 mit einem simplen FORTRAN-Compiler und einem basischen Betriebssystem an den Kunden ausgeliefert. Aufgrund der innovative Natur der Maschine traten ein Jahr später mehrere **Probleme** auf. Einige von ihnen waren auf den Herstellungsprozess und fehlerhafte Speichereinheiten zurückzuführen. Andere Hauptbeschwerden betrafen das enthaltene Betriebssystem sowie den Mangel an virtuellem Speicher und Interrupts auf den PPUs [43][44].

Allgemein war der CDC 6600 **für ihrer Zeit eine sehr leistungsstarke Maschine** und brachte die Technologie voran. Der Erfolg führte zur Veröffentlichung mehrerer Versionen (d.h. Upgrades) wie dem CDC 7600. Das Kühlsystem, die Verwendung von Silizium, der Anzeigetafel-Mechanismus („Scoreboard“) und die Verwendung von mehr als einer der gleichen parallelen Funktionseinheiten (Figur 2) waren wegweisende Entwicklungen. Dies war der notwendige Übergang von skalaren zu superskalaren Prozessoren und **motivier- te die Forscher und Unternehmen die Parallelität auf Befehlsebene weiter zu entwickeln**. Die Konkurrenz brauchte Jahre, um einen Computer anzupassen und herauszubringen, der mit seiner Leistung mithalten könnte.

Tatsächlich nahm IBM das ganz persönlich und fühlte sich verspottet, als sie beobachtete, wie ein so kleines Unternehmen wie Control Data Corporation das geschafft hatte. Der damalige CEO IBMs schrieb ein kurzes aber unvergessliches Brief an den Mitarbeitern: [47]

*„Last week CDC had a press conference during which they officially announced their 6600 system. I understand that in the laboratory developing this system there are only 34 people, “including the janitor.” Of these, 14 are engineers and 4 are programmers, and only one has a Ph. D., a relatively ju-*

*nior programmer. To the outsider, the laboratory appeared to be cost conscious, hard working and highly motivated. Contrasting this modest effort with our own vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world’s most powerful computer. At Jenny Lake, I think top priority should be given to a discussion as to what we are doing wrong and how we should go about changing it immediately.“*

Der CDC 6600 war in den kommenden Jahren, auch wenn er nicht perfekt war, der Baustein für Generationen von Computern, und auch eine Motivation und Inspiration für so eine wichtige Firma wie IBM.



Figure 4: CDC 6600 (Bild: Computer History Museum)

### 2.1.2 Entwicklung und Beiträge des IBM System/360 Model 91

Der Erfolg des CDC 6600 war ein schwerer Schlag für



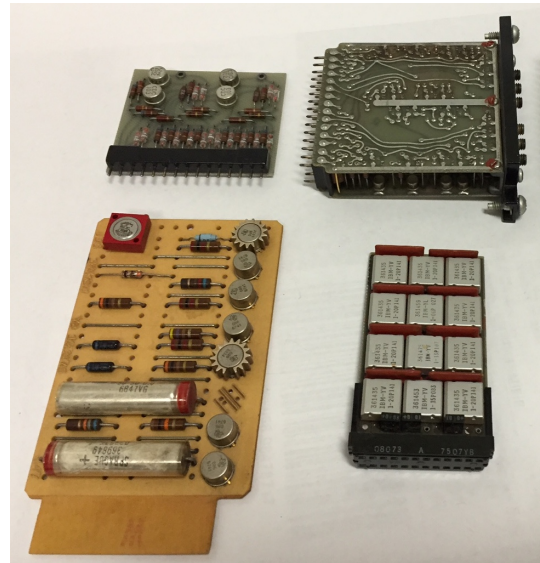
IBM. Seine Leistung war 3 mal besser als die vom IBM 7030 „Stretch“. Das CDC-Team war sich der Spezifikationen dieses Computers bewusst, was zum Teil darauf zurückzuführen war, dass Thornton an dem Seminar teilgenommen hatte, auf dem es besprochen wurde, und dies verschaffte ihnen einen Vorsprung. Die Zeit war jetzt reif für IBM, etwas dagegen zu unternehmen.

### *Neue Familie, neue Technologie*

Am 7. April 1964, nachdem der CDC 6600 veröffentlicht wurde, war ihr neuer Plan in Gang gesetzt: Die Entwicklung einer **neuen Produktfamilie**, IBM System/360. Diese Familie würde jede Produktlinie ersetzen, die es derzeit auf dem Markt gab: Die wissenschaftlichen Rechner wie IBM 7090, die für kleine Unternehmen wie IBM 1401 und die für größere Unternehmen wie IBM 7080. [13] Die neue Familie würde eine **neue ISA haben und wäre daher mit Vorgängermodellen nicht kompatibel**. Dies war ein notwendiger, aber gleichzeitig sehr mutiger Schritt. Um innovativ zu sein, muss manchmal die Abwärtskompatibilität für die Leistung geopfert werden [12]. Gleichzeitig würden sie sicherlich einige Kunden an CDC und andere Konkurrenten verlieren, sobald sie ein Upgrade durchführen müssten. Das Positive daran war jedoch, dass sie trotz des Verlusts der Abwärtskompatibilität noch von den vorherigen Maschinen gelernt hatten und viele ihrer einzigartigen Eigenschaften und Innovationen (sogar die der Konkurrenz) verbessert und auf die IBM System/360 Familie übertragen wurden.

Insbesondere das Model 91 hatte mindestens drei IBM-Computern als Inspiration. Der IBM 7030 „Stretch“ (der aus dem Seminar) hatte Pipelining und hatte auch Verzweigungsvorhersage, die für ihre Zeit bahnbrechend, aber auch experimentell war. Das Wiederherstellen des Zustandes nach einen falsch vorhergesagten Zweigs hat beispielsweise die Leistung der Maschine beeinträchtigt und ist einer der Gründe, warum dies kein kommerzieller Erfolg war. Der IBM 7090 hatte kein Pipelining, es war aber auch innovativ, da er anstatt Vakuumröhren ECL-Schaltkreisen benutzt hat, so wie das Modulsystem der CDC-Computer (Figur 5). Dadurch war es fünfmal schneller als sein Vorgänger, der auch viele Neuerungen wie Gleitkomma-Anweisungen und Indexregister auf den Tisch gebracht hatte. Eine weitere große Inspirations- und Motivationsquelle war natürlich der CDC 6600 der Konkurrenz, der beispielsweise ILP und seine Cordwood-Module nutzte [13] [12].

Das Ziel des Model 91 war es, die höchstmögliche Leistung für die Technologie zu erzielen, auf der die System/360-Familie basierte. Es **musste sich im Bereich des wissenschaftlichen Rechnens mit der CDC 6600 messen** und richtete sich an Kunden wie das US-Verteidigungsministerium, die NASA oder die US-Atomenergiekommission sowie andere Forschungseinrichtungen und Universitäten. Der angestrebte CPI war immer noch 1. Das Team erreichte diese Ziele durch Deep Pipelining (20 *stages*); schnelle, gepufferte und gleichzeitige Speicherzugriffe; Wasserkühlen; innovative Implementierungen der Gleitkommaoperationseinheiten (Divisionseinheit verwendet den brandneuen Goldschmidt-Algorithmus); und eine effizientere Verzweigungsvorhersage (*branch-loop*-Modus: Wenn das Ziel in der Nähe war, wurde immer „springt“ vorhergesagt). Obwohl letzterer besser war als der auf IBM SStretchimplementierte, stark verzweigte, nicht wissenschaftlicher Code funktionierte immer noch schlechter (3+ CPI). [13]



**Figure 5: Die von CDC bzw. IBM benutzte Technologie für Packaging (Bild: M. Smotherman, Clemton University Lecture Notes)**

Ein weiteres sehr respektables Ziel des Teams war die Vereinfachung der Wartung, Fehlerprüfung und Fehlerbehebung: Alle Datenübertragungen und Rechenoperationen wurden überprüft, es gab ein Logging-system, und man konnte sogar den Status jedes Registers auf der Konsole oder mit blinkenden Lichtern anzeigen. Laut Flynn war ein 15-20% des Prozessors für diese Funktionen reserviert und wir sollten uns noch heute mehr darauf konzentrieren. Dass das System aufgrund von Softwarefehlern abstürzt, solle inakzeptabel sein und sei ein Misserfolg für die Software und auch für die Hardware-Entwickler, wenn sie nicht die erforderlichen Techniken zur Gewährleistung der Zuverlässigkeit implementieren, die seit Jahrzehnten verfügbar sind [12].

Das Entwicklungsteam meinte, man könne mit der Technologie nur so weit kommen [2]. Thornton hatte das auch im Bezug auf den CDC 6600 und seine Innovationen gedacht [43]. Abgesehen von den oben erwähnten architektonischen und meist technologischen Fortschritten (wie kürzere Speicherzugriffszeiten, Pufferung, Pipelining usw.) gab es eine sehr wichtige Entwicklung im IBM System/360 Model 91, die diesen Computer tatsächlich auszeichnete, und das war im Bereich Parallelität auf Befehlsebene: **Tomasulos Algorithmus** [2] [46]. Es wurde von Robert Tomasulo während seiner Zeit bei IBM entwickelt. Es ermöglicht eine weitaus **effizientere Verwendung mehrerer Funktionseinheiten** durch die Ausführung von Befehlen in einer anderen, in der Zeitpunkt gemäßigtere Reihenfolge (die sogenannte „out-of-order execution“). Die Schwierigkeit hier ist die Abhängigkeiten zu respektieren. Im Fall des Model 91 wurde es **nur auf Gleitkomma-Operationseinheiten angewendet**.

### *ILP und Tomasulos Algorithmus: der wichtigste Beitrag des Model 91*

Gleitkommaoperationen wurden auf dem IBM System/360 Model 91 aufgrund mehrerer Neuerungen bereits besser ausgeführt, wie z.B. bessere Implementierungen der Multiplika-

tion (sie dauerte drei Zyklen) [2] und der Division. Die einfachste Möglichkeit, die Leistung in diesem Zusammenhang zu verbessern, bestand darin, Festkomma- und Gleitkommaeinheiten zu trennen, damit die verschiedenen Arten von Operationen parallel ausgeführt werden können. Wie bereits erwähnt, stützen sich wissenschaftliche Anwendungen fast ausschließlich auf Gleitkommaoperationen, deswegen würde die Verbesserung in diesem Zusammenhang minimal. Was wir brauchen ist Parallelität zwischen Gleitkommaoperationen auch. Erstens ist es besser die ALU in verschiedene Einheiten zu trennen: Die Durchführung einer Multiplikation in einer ALU, die sowohl Summen als auch Multiplikationen ausführen kann, erfordert mehr Zyklen als die Verwendung einer spezialisierten Einheit. Auf diese Weise **können z.B. Summen und Multiplikationen gleichzeitig ausgeführt werden, oder z.B. zwei Summen, wenn es zwei Addierer gibt. Dies ist es, was einen superskalaren Prozessor auszeichnet.** Es ergibt sich jedoch ein neues Problem: Die einzige Unterscheidung zwischen Gleitkomma- und Festkommaoperationen war, dass sie unabhängig waren, da Gleitkommaoperationen keine Ganzzahlregister verwenden und umgekehrt, deswegen könnten sie ohne Probleme in einer anderen Reihenfolge ausgeführt werden. **Abhängigkeiten zwischen Gleitkommaoperationen müssen jetzt behandelt werden**, wenn mehrere Befehle desselben Typs parallel ausgeführt werden sollen, **und hier kommt die *Out-of-Order Execution* durch Tomasulos Algorithmus ins Spiel [46].**

Erstens, um das Prozess zu vereinfachen werden die Register neu benannt in einem Verfahren namens „register renaming“. So denkt der Computer, dass es mehr Register zu Verfügung hat. So verschwinden auch die Gegen- und Antidependenzen im Programm. Der wichtigste Teil der OoO-Ausführung sind aber wahrscheinlich Reservierungsstationen: Stellen, an denen Operationen warten können, bis die Funktionseinheit und die Daten bereit sind, um sie ausführen zu können. Zum Beispiel, da der CDC 6600 keine OoO-Ausführung machen könnte wurden diese Einheiten blockiert als die Operanden nicht verfügbar waren. So wird die Einheit nicht blockiert, und kann währenddessen weitere Operationen ausführen. Ein sogenanntes Common Data Bus (CDB) verbindet die Register mit den Reservierungsstationen jeder Einheit. Durch das CDB senden sich alle Komponente des Prozessors die Ergebnisse und Operanden,

die sie für die Ausführung brauchen. Da es sich von einer Art Broadcast-Kommunikation handelt weiß der Empfänger, dass eine Nachricht für ihn ist, weil sie markiert (*tagged*) sind. Wenn Tomasulos Algorithmus angewendet wird, um eine korrekte „out-of-order execution“ zu verführen, muss jeder neue Befehl mit allen anderen Befehlen, die gerade ausgeführt werden verglichen. Wenn es unabhängig ist und die benötigte Einheit und die Daten verfügbar sind, kann es sofort ausgeführt werden. Wenn nicht, wird an die Reservierungsstation der benötigten Einheit weitergeleitet, die eine Art Warteschlange ist. Sobald die Einheit freigegeben ist und die Operanden bereit sind, wird der nächste Befehl von der Reservierungsstation in der Einheit ausgeführt. Wenn das Ergebnis fertig ist, geht es überall dort hin, wo es die CDB durchlaufen muss, sei es ein Register oder eine andere Reservierungsstation, wo ein Befehl auf das Ergebnis wartet.

Um es einfacher zu erklären, so würde eine umgangssprachliche Version vom Algorithmus aussehen:

- **Neues Befehl wird ausgegeben. Bedingung: Ist eine Reservierungsstation verfügbar?**

Ja: Renaming im Register. Operation, Operanden und/oder (*tag*) werden zur Station weitergeleitet.

Nein: Stall-Zyklus

- **In Reservierungsstationen warten Befehle an fehlende Operanden, die über dem CDB von einem Register oder einer Einheit ankommen. Bedingung: Ist die Marke (*tag*) der Nachricht == meine Marke?**

Ja: Ich wartete auf dieser Operand und muss es speichern.

- **Bedingung: Sind beide Operanden schon in der Reservierungsstation?**

Ja: Befehl wird zur Ausführungseinheit weitergeleitet.

- **In der Ausführungseinheit. Bedingung: Ist die Operation fertig?**

Ja: Broadcast mit Marke über CDB.

- **Die Register warten auf Nachrichten, die über dem CDB ankommen. Bedingung: Ist die Marke (*tag*) der Nachricht == meine Marke?**

Nein: Ein späteres Befehl wird den neuesten Wert später senden, man muss mehr warten.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
i1	mul r3, r1, r2	IF	ID	M1	M2	M3	M4	M5	M6	WB																
i2	add r5, r3, r4		IF	ID	-	-	-	-	-	A1	A2	A3	A4	WB												
i3	add r7, r2, r6			IF	-	-	-	-	-	ID	A1	A2	A3	A4	WB											
i4	add r10, r8, r9									IF	ID	A1	A2	A3	A4	WB										
i5	mul r11, r7, r10										IF	ID	-	-	-	M1	M2	M3	M4	M5	M6	WB				
i6	add r5, r5, r11											IF	ID	-	-	-	-	-	-	-	-	A1	A2	A3	A4	WB

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
i1	mul r3, r1, r2	IF	ID	M1	M2	M3	M4	M5	M6	WB																
i2	add r5, r3, r4		IF	ID	-	-	-	-	-	A1	A2	A3	A4	WB												
i3	add r7, r2, r6			IF	ID	A1	A2	A3	A4	-	-	-	-	-	WB											
i4	add r10, r8, r9				IF	ID	A1	A2	A3	A4	-	-	-	-	-	WB										
i5	mul r11, r7, r10					IF	ID	-	-	-	M1	M2	M3	M4	M5	M6	WB									
i6	add r5, r5, r11						IF	ID	-	-	-	-	-	-	-	-	A1	A2	A3	A4	WB					

Figure 6: Programmablaufdiagramm. Ohne und mit OoO-Ausführung. [24]

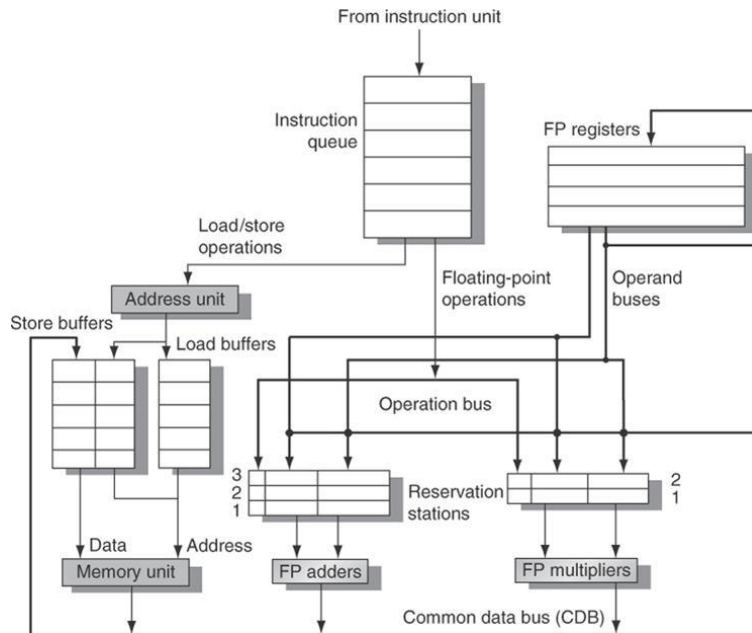


Figure 7: Tomasulos Algorithmus implementiert im IBM System/360 Model 91 [46]

Ja: Ich wartete auf dieses Ergebnis und muss es speichern.

Es ist aber einfacher zu verstehen mit einem Beispiel: In der Figur 6 benötigt dasselbe Programm mit einer „out-of-order execution“ 5 Zyklen weniger. Dies ist auf die Tatsache zurückzuführen, dass zum Beispiel, normalerweise während i2 wartet bis i1 fertig ist, um sein Ergebnis zu erhalten, keine der folgenden Befehle diese 5 „stall-Zyklen“ ausnutzen kann. Bei einer „out-of-order execution“ führt jede der folgenden Befehle (von i3 bis i6) in diesem Zeitraum etwas aus, zumindest das Abrufen und Dekodieren und bei i3 und i4 sogar die Ausführung selbst.

Zyklus für Zyklus ist es bis Zyklus 3 für beide Methoden gleich, aber was dann passiert, ist ganz anders: Ohne OoO-Ausführung wird i2 dekodiert und wartet in der Einheit, bis der benötigte Operand bereit ist. So können andere folgende Befehle die Einheit nicht benutzen. **Bei OoO-Ausführung wartet es an der Reservierungsstation und die Einheit wird nicht blockiert.** Andere Befehle können die Einheiten in der Zwischenzeit verwenden. Im Zyklus 9 ist i1 fertig und das Ergebnis wird dorthin weitergegeben, wo es benötigt wird. In diesem Fall braucht i3 das Ergebnis, und deswegen, anstatt ein Register als Ziel zu haben, geht es zur Reservierungsstation wo i2 wartet um sein eigenes Ergebnis zu berechnen. Und so geht es immer weiter.

### Rezeption und Schlussfolgerungen

Tomasulos Algorithmus und viele darauf basierende Variationen wurden in vielen späteren Prozessoren implementiert, insbesondere in den neunziger Jahren und später. Die spätere Verwendung von Caches, die Tatsache, dass der Algorithmus nicht auf eine einzelne Art von Pipeline-Struktur beschränkt war, und seine Toleranz gegenüber Verzweigungsvorhersagen erleichterten dies. Alle diese Funktionen waren für neuere superskalare Prozessoren, die mehrere Befehle pro Zyklus ausgaben, attraktiv. Darüber hinaus wurde es



Figure 8: Der IBM System/360 Model 91 im NASA Greenbelt Space Flight Center (Bild: IBM Photo Archive)

im IBM System/360 Model 91 nur im Zusammenhang mit Gleitkommaoperationen angewendet, aber später verwendeten Computer es bei der Planung von Befehlen aller Art, von Ganzzahloperationen bis hin zu Lade- und Speicheroperationen.

Der IBM System/360 Model 91 war **allgemein kein sehr erfolgreicher Rechner** (nur ungefähr 15-20 wurden produziert). Die meisten hat die NASA gekauft und andere hat IBM gehalten um sie selber zu benutzen. In Bezug auf das Upgrade (Model 95) wurden nur zwei Exemplare hergestellt, und ihr einziger Kunde war erneut die Raumfahrtbehörde [29]. Der Model 91 war aber **ein notwendiges Opfer für IBM und den ganzen Absatzmarkt**. Auch die Nachfolger, die auf den selben Entwicklungen basiert sind, waren sehr erfolgreich. Seine Relevanz ist historisch: Er wird aufgrund Tomasulos Algorithmus in Kursen zum Thema Rechnerarchitektur immer erwähnt. Alles in allem war er **für seine Zeit ein wirklich leistungsstarker Computer**. Es war **der erste, der eine OoO-Ausführung implementierte**, und kann als den ersten echt superskalaren Com-



puter betrachtet werden, obwohl er auch nicht mehr als eine Anweisung pro Zyklus ausgegeben hat, da es noch auch einen IPC-Wert von 1 anstrebte. Sowohl dieser Computer als auch der CDC 6600 haben **einen großen Beitrag zur Technologie und zur Rechnerarchitektur** geleistet und mit ihren Innovationen die Grenzen der Computerindustrie erweitert. Diese beiden „Väter der Superskalarität“ sind die **repräsentativsten Computer in der frühen Geschichte der Parallelität auf Befehlsebene**, und alle späteren Modelle (auch die aus der Konkurrenz) basierten darauf. Um Flynn, der berühmte Computerarchitekt, zu quotieren, wenn die Industrie jetzt so tapfer wäre mutige Innovationen zu machen wie damals, könne die gegenwärtige Leistung von Computern ganz anders aussehen, da sie weiterentwickelt worden wären [12].

## 2.2 Entwicklung zu den modernen Mikroprozessoren

### 2.2.1 Das IBM Advanced Computer System

Neben Projekt X das zum IBM S/360 Model 91 führte und die Leistung des Stretch Computers um den Faktor 10 bis 30 übersteigen sollte, gab es auch das wesentlich engagiertere Projekt Y[42]. Das Ziel war ein Schub um ein 100-faches an Leistung. Das Projekt wurde dem IBM Watson Research Center zugeteilt. Da das ACS Projekt inkompatibel zur S/360 Architektur war fehlte es ihr an Unterstützung innerhalb IBMs. Die meisten geplanten Projekte sollten S/360 kompatibel sein, weswegen später auch im ACS Projekt daran gearbeitet wurde. Es gab ein internen Konflikt über zwei Designs. Das IBM Management entschied sich, dem AEC/360 Design den Vorrang zu geben. Durch die Neuorientierung verließ ein großer Teil des Designteams das Projekt und das Projekt wurde ACS/360 benannt. Der Projektleiter stellte dem Management drei Modelle vor um mit dem ACS/360 Gewinn zu erzielen. 1969 wurden die Modelle allerdings abgelehnt und das Projekt eingestellt.

### 2.2.2 Zwei Jahrzehnte ILP Forschung

Anfang der siebziger Jahre wurden einige Studien veröffentlicht, die sich mit der Parallelisierung auf dem Befehlslevel befassen.

Garold Tjaden und Michael Flynn untersuchten, wie viele Instruktionen simultan dekodiert und an die „Execution Stage“ weitergeleitet werden konnten[45]. Sie beschrieben ein Verfahren zur Detektion und Anordnung von Abhängigkeiten in einem Befehlsstrom. Zur Evaluierung verwendeten sie einen Simulator, der nahelegte es wäre eine durchschnittliche Leistungssteigerung von 86% möglich.

Ed Riseman und Caxton Foster publizierten 1972 ebenfalls ein Papier auf dieser Ebene[14]. In diesem legten sie ihre Untersuchungen zur parallelen Verteilung und Ausführung in „Issue Stage“ und „Execution Stage“ dar. Sie kamen zu dem Ergebnis, dass zwar ein gewisses Potential für die Parallelisierung zwischen konditionalen Sprüngen auf damaligen Rechnern vorhanden war, dieses aber limitiert wäre. In ihren Tests kamen sie auf eine mögliche Leistungssteigerung um 72%.

Diese Arbeiten sind ein gutes Beispiel für die damalige Skepsis, die bezüglich der Ausnutzung von Parallelisierung durch die Anwendung superskalarer Techniken vorherrschte.

Bis in die Mitte der 1980er Jahre wurden dementsprechend auch kein Rechner mit superskalarer Architektur vermarktet[36].

In den folgenden Jahren wurden unterschiedliche Methoden und Ideen zur Parallelisierung ausgearbeitet. Es wurden sowohl Vektorprozessoren als auch Multiprozessoren entwickelt. Ebenfalls gab es die Idee mehrere kompatible Instruktionen zu verschmelzen und sie gemeinsam durch die Pipeline zu leiten. Vertreter dieser Techniken sind unter anderem der CRISP Mikroprozessor[3] und später der Transputer T9000[20] sowie der TI SuperSPARC[26] anfang der neunziger Jahre.

Wichtige Vertreter der Vektorprozessoren waren der CDC Star-100[30], der TI ASC[6] und der Cray-1[33], welcher von vielen als der erste Supercomputer angesehen wird. Einen großen Vorteil hatten diese Prozessoren vor allem bei wissenschaftlichen und numerischen Programmen. Oft musste dieselbe Instruktion für viele Daten verwendet werden. Durch Vektorprozessoren war es möglich, eine große Menge Daten gleichzeitig dieselbe Operation ausführen zu lassen. Das ist sowohl schnell als auch effizient. Auch heutzutage werden Vektorprozessoren für spezielle Aufgaben eingesetzt und bieten Vorteile gegenüber anderen Architekturen[18]. Viele Heimcomputer haben sogar eigene Vektor- und Multimediaeinheiten, um in entsprechenden Anwendungen gute Leistungen zu erzielen. Aus Gründen, welche in Sektion 2.3 beschrieben sind, ist vor allem Energieeffizienz ein primärer Designfaktor. Durch die Abarbeitung vieler Daten in einem Schub, ist im Gesamten der Stromverbrauch wesentlich geringer.

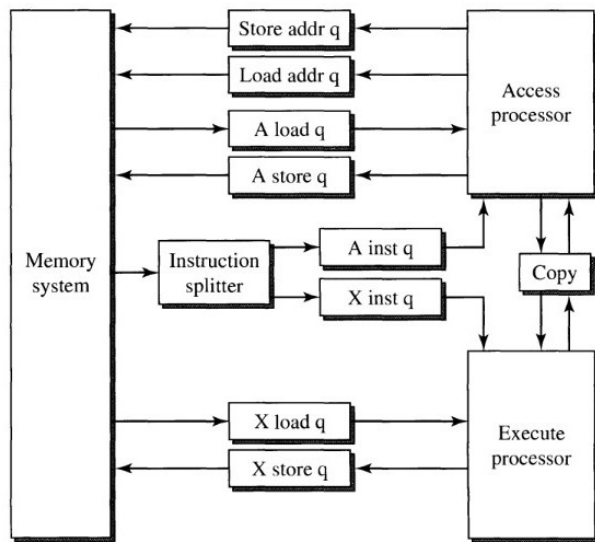
### VLIW Architekturen.

Weiterhin wurden anfang der achtziger Jahre die sogenannten „Very-Long-Instruction-Word“ (VLIW) Architekturen vorgestellt[11, 32]. Diese unterschieden sich in ihrer Herangehensweise über die neue Positionierung des Dynamic-Static Interface. Das DSI gibt an, welche Aufgaben statisch bei der Kompilierung oder dynamisch bei der Ausführung des Programms ausgeführt werden. Durch Verschiebung des DSI in Richtung statischer Programmausführung sollte die Anordnung der Befehle bereits während der Kompilierung durchgeführt werden, um diese Aufgabe bei der zeitkritischen Ausführung auf der Hardware zu umgehen. Umgesetzt wird dies bei VLIW-basierten Architekturen indem man den Compiler unabhängige Befehle finden lässt die parallel ausgeführt werden können, die also keine Abhängigkeiten untereinander haben und mehrere dieser Operationen in einer sehr langen Instruktion bündelt. Die Operationen dieses Bündels können dann nach dem Dekodieren des VLIW-Befehls auf der Hardware parallel auf mehrere Funktionseinheiten verteilt werden. Auch modernere Architekturen basieren auf der Idee von VLIW. Intel hatte vor einigen Jahren die Itanium Serie[35] auf den Markt gebracht, welche die Grundidee von VLIW übernahm und weiter ausbaute. Auch die TRIPS Architektur[34] hat ihren Ursprung ebenfalls der Idee von VLIW zu verdanken.

### Decoupled Access/Execute Architekturen.

Anfang der achtziger Jahre publizierte James Smith mehrere Arbeiten über sogenannte „Decoupled Access/Execute“ Architekturen[37, 38]. In diesen Designs wurde die Abarbeitung von Speicherbefehlen und numerischen Berechnungen voneinander entkoppelt, um möglichst früh Speicherzugriffe

durchführen zu können.



**Figure 9: Astronautics ZS-1 Blockdiagramm (Bild: Modern Processor Design[36])**

Abbildung 9 zeigt den Astronautics ZS-1 der aus den Arbeiten von James Smith resultierte. Die Funktionsweise wird im Nachhinein grob geschildert.[36, 39, 40]

Der „Instruction Splitter“ war direkt mit dem Speicher verbunden und konnte bis zu zwei 32-bit oder eine 64-bit Instruktion abrufen. Verzweigungen wurden ebenfalls hier ausgeführt und dann vom Instruktionsstrom entfernt. Die Befehle wurden anschließend in Warteschlangen gelegt. Die „A“ Befehlswarteschlange besaß Platz für bis zu vier Einträge, die „X“ Befehlswarteschlange konnte dagegen bis zu 24 Befehle speichern. Die Befehle wurden anschließend in sequentieller Reihenfolge abgefertigt.

Der „Execute Processor“ beinhaltete eine ALU für Integer, eine Shift-Einheit und eine Multiplikation/Divisions-Einheit. Der „Access Processor“ nutzte eine Logikeinheit, einen Fließkommaaddierer, einen Fließkommamultiplizierer und eine gegenseitige Approximationseinheit.

Der Astronautics ZS-1 sowie der Culler-7, welcher 1986 angekündigt wurde, sind unabhängig voneinander entworfen worden und waren die ersten Prozessor, welche mehrere Instruktionen aus einem Instruktionsstrom laden, dekodieren und zur Ausführung weiterleiten konnten. Die Idee, Integer Instruktionen sowie Speicherzugriffe in einer eigenen Pipeline von den Fließkommaberechnungen zu trennen wurde später von vielen Architekturen aufgegriffen.

DAE Architekturen sind generell nicht unbedingt als Universalrechner geeignet, da die meisten Programme viele Verzweigungen implementieren. Entsprechende Untersuchungen liefert[19].

### Entwicklungen der achtziger Jahre bei IBM.

Wie J.P.Shen und M.H.Lipasti in [36] beschreiben, startete IBM das Cheetah-Projekt anfang der achtziger Jahre, in welchem einige funktionale Einheiten entwickelt wurden, welche später im Design der RS/6000 Rechner zur Anwendung kamen. Tilak Agerwala, der dem Projekt vorsah führ-

te erstmals den Begriff der superskalaren Rechner ein, um Architekturen zu beschreiben die, wie der ACS und das Cheetah-Projekt, mehrere Instruktionen parallel laden und dekodieren konnten.

Das Design wurde dann 1985/86 weiterentwickelt und resultierte im America Design, welches vier Instruktionen simultan dekodieren konnte. Anschließend wurde das Design im „IBM Austin Development Lab“ verfeinert um die RS/6000 Architektur zu entwickeln, die auch unter dem Namen „RIOS“ oder „POWER“ bekannt sind.

### HPS Forschung.

Mitte der 80er Jahre brachte Yale Patt mit seinen Studenten an der University of California eine Arbeit[27] ein, welche eine verallgemeinerte Version des Tomasulo Algorithmus nutzte, um einen sequentiellen Instruktionsstrom in einen partiellen Datenflussgraphen zu konvertieren. Komplexe CISC Instruktionen wurden dabei zu RISC-ähnlichen Mikrooperationen übersetzt, welche die Knoten des Graphen darstellen. Patt und seine Studenten konnten dabei die Anzahl der nötigen CPI für einen VAX Befehl von sechs auf zwei Zyklen verringern. Die Übersetzung von CISC Instruktionen in Mikrooperationen wird heutzutage in fast allen IA32 Prozessoren eingesetzt (siehe Sektion 2.2.4).

### 2.2.3 Verbreitung superskalarer Architekturen

Ende der achtziger Jahre und anfang der neunziger Jahre wurden zahlreiche superskalare Architekturen entwickelt. IBM startete mit den RS/6000 Rechnern (später POWER) und Intel veröffentlichte die i660 Architektur, die im ersten superskalaren Mikroprozessor genutzt wurde. Diese beiden Architekturen werden, entsprechend der Ausführungen in [36], in dieser Sektion vorgestellt.

### IBM POWER (RIOS).

1989 kündigte IBM mit der RISC System 6000 Familie die ersten Vertreter der POWER Architektur an. Die RS/6000 Computer waren die ersten Workstations und Server mit superskalarem Design.

Die Architektur basierte auf den Ergebnissen des America Prozessor Designs. Eine Besonderheit war die Ausführung einer Instruktion aus verschmolzenen Multiplikations- und Additionsbefehlen. Diese Instruktion hatte vier Operanden, wovon die ersten beiden multipliziert und das Ergebnis anschließend mit dem dritten Operanden addiert wurden. Diese Berechnung ist zwar sehr speziell, wurde allerdings in den meisten numerischen Programmen oft verwendet und trug einen entsprechend großen Teil zur Leistung der Architektur bei. Weiterhin wurde die Ausführung auf drei separate Funktionseinheiten mit eigenem Registerset aufgeteilt. Dadurch reduziert sich die Komplexität, da zwischen Funktionseinheiten nicht auf Abhängigkeiten getestet werden musste.

Im Laufe der Zeit wurden einige Verbesserungen und Erweiterungen zur POWER Architektur hinzugefügt. 1991 suchten Apple und Motorola eine neue Plattform für ihre Architekturen und schloßen sich mit IBM zur AIM Allianz zusammen. Zusammen entwickelten sie die PowerPC ISA (ISA steht für: Instruction Set Architecture), die stark an der POWER Architektur orientiert aber durch Einbringen von Apple und Motorola erweitert wurde. Die PowerPC ISA war eine RISC Architektur mit einem Befehlssatz der sowohl 32-bit als auch 64-bit unterstützte. Die Architektur sollte sowohl im leistungsschwachen Segment wie einge-

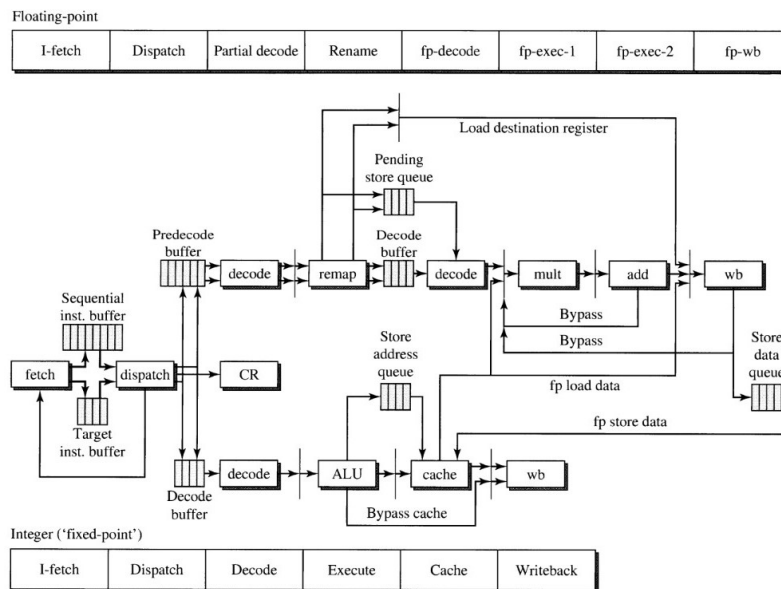


Figure 10: IBM Power (RIOS) Pipeline Stages (Bild: Modern Processor Design[36])

bettete Mikrocontroller, als auch im leistungsstarken Spektrum wie bei Supercomputern, Servern und Workstations eingesetzt werden. Später wurden im Amazon Projekt die PowerPC-AS Erweiterungen hinzugefügt um die Prozessoren mit den AS/400 Systemen kompatibel zu machen. Die PowerPC ISA wurde später durch die POWER ISA ersetzt, die von der 2004 gegründeten Organisation Power.org und später der OpenPOWER Foundation entwickelt und erweitert wurden. Die aktuellen Mikroprozessoren der POWER Reihe nutzen die Power ISA 3.0 mit den Prozessoren der POWER9 Generation.

Abbildung 10 zeigt die RIOS Pipeline, die erste Implementierung der POWER Architektur. Im Folgenden werden die Abarbeitungsschritte von Instruktionen in den unterschiedlichen Phasen besprochen.

Der Instruktionscache und die Verzweigungseinheit konnten bis zu vier Befehle laden. Bei sequentieller Ausführung wurden die Befehle im acht Felder breiten „Sequential Instruction Buffer“ gespeichert. Die Verzweigungseinheit überwachte die ersten fünf Einträge und suchte nach Sprungbefehlen. Sobald eine Verzweigung erkannt wurde, lud die Verzweigungseinheit die ersten vier Instruktionen an der Zieladresse des Sprunges, um sie im „Target Instruction Buffer“ zu speichern. Sollte der Sprung wirklich durchgeführt werden, wurde der Inhalt des „Target Instruction Buffers“ an den „Sequential Instruction Buffer“ übermittelt. Anschließend wurden die Register auf den Zustand vor Ausführung des Sprunges gesetzt und die neuen Instruktionen ausgeführt. Sollte der Sprung nicht zustande kommen, wurde einfach der Inhalt des „Target Instruction Buffers“ gelöscht und die Ausführung normal fortgesetzt. Es konnten jeweils zwei Befehle direkt zur Verzweigungseinheit und den Puffern für die Funktionseinheiten befördert werden. Dabei konnten sowohl Integer als auch Fließkommabefehle abgefertigt werden. Während die Verzweigungseinheit Sprungbefehle ausführte, gab es ab der „Execute Phase“ jeweils eine Pipeline für Integer (oder bei IBM auch „fixed-point“ genannt) und Fließkommabefehle. „LOAD“ und „STORE“ Befehle nutz-

ten beide Pipelines, eine für die Berechnung der Adressen und eine für die geplante Operation.

In der Integer-Pipeline wurden die Befehle dekodiert und daraufhin an die ALU gesendet. Nach der Berechnung wurden sie anschließend sowohl an die Cache-Phase übergeben, als auch an die Writeback-Phase (in der Abbildung „wb“). „STORE“ Befehle, welche die Integer-Pipeline für die Adressgenerierung verwenden, wurden in der „Store Address Queue“ abgelegt, bis die Daten aus der Fließkomma-Pipeline verfügbar sind. „LOAD“ Befehle wurden sowohl an die Writeback-Phase als auch an die „Execute Phase“ übergeben, um Wartezeiten zu vermeiden.

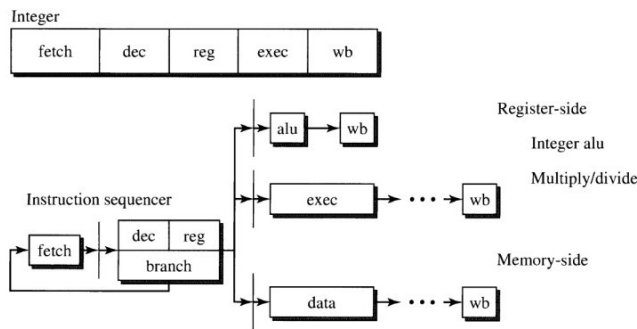
Die Fließkomma-Pipeline dekodierte die Befehle teilweise um „LOAD“, „STORE“ und ALU Operationen zu identifizieren. In der „Remap Stage“ wurden durch „Register Renaming“ 38 physische Register auf die 32 architektonischen Register abgebildet. Anschließend wurden die Befehle an die entsprechenden Phasen für die jeweiligen Befehlstypen weitergeleitet. Die Operationen wurden dekodiert, Befehle gelesen und die Berechnungen durchgeführt, bevor sie in der Writeback-Phase landeten.

### Intel i960CA.

Basierend auf der 1988 veröffentlichten i960 RISC Architektur vermarktete Intel den unter Glenn Hinton und Frank Smith entwickelten i960 CA als ersten superskalaren Mikroprozessor. Er enthielt einen 1K-byte Zwei-wege set-assoziativen Instruktionscache. Dadurch konnten bis zu zwei Speicherbänke parallel geladen werden. Außerdem wurden spekulativ Daten geladen, falls eine Verzweigung ausgeführt werden sollte, um Verzögerungen durch einen Cache-Miss vorzubeugen. Die Erholung von fälschlicherweise ausgeführten Sprungbefehlen musste entweder softwareseitig oder durch eine spezielle Instruktion behandelt werden.

Als Teil des „Instruction Sequencers“ besaß der Prozessor ein einzelnes Registerset mit sechs Anschlüssen, jeweils drei für die „Register Execution Unit“ und die „Memory Execu-

tion Unit“. Ebenfalls im „Instruction Sequencer“ enthalten, waren eine „Instruction Fetch Unit“, der „Instruction Cache“ und ein „Parallel Instruction Scheduler“. Die Pipeline Phasen sind in Abbildung 11 zu sehen und werden im Folgenden kurz erläutert.



**Figure 11: Intel i960CA Pipeline Stages (Bild: Modern Processor Design[36])**

Die „Instruction Fetch Unit“ konnte alle zwei Zyklen vier Instruktionen laden. Waren die Befehle geladen, gab der „Parallel Instruction Scheduler“ bis zu zwei Befehle jeden Zyklus an die „Execution Units“ weiter, während er zusätzlich einen Verzweigungsbefehl ausführen konnte. Er konnte also maximal drei Instruktionen in einem Zyklus verarbeiten. Allerdings wurden erst neue Befehle verarbeitet, wenn alle vorherigen weitergeleitet wurden. Dadurch wurden auf längere Dauer gesehen, maximal zwei Instruktionen pro Zyklus ausgeführt. Durch „Register Scoreboarding“ wurde festgestellt, ob ein Register bereits von einer Operation besetzt wurde oder ob der Befehl unabhängig war. Anschließend konnte er je nach Befehlstyp zur entsprechenden „Execution Unit“ befördert werden. Die „Register Execution Unit“ führte ausschließlich Instruktionen aus, welche auf Registern arbeiteten. Die „Memory Execution Unit“ führte dagegen ausschließlich Speicherbefehle wie „LOAD“ und „STORE“ aus.

Nachdem die Berechnungen der jeweiligen Instruktion abgeschlossen waren, wurden sie in der Writeback Phase übernommen.

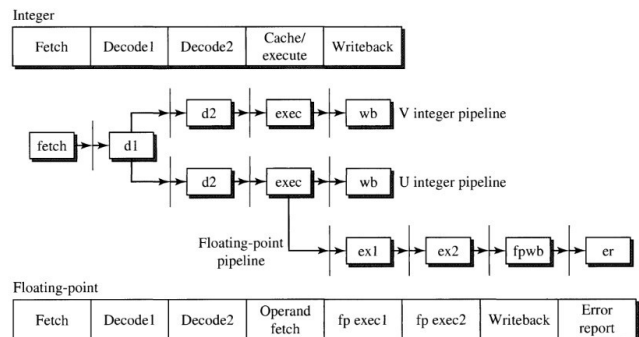
Detailliertere Informationen über die i960CA Architektur sind auffindbar in [21] und [36].

#### 2.2.4 Entwicklung der modernen, superskalaren Mikroprozessoren

Mikroprozessoren haben sich schnell als die am weitesten verbreiteten Prozessortypen weltweit entwickelt. Dabei haben auch superskalare Architekturen eine entscheidende Rolle gespielt. Heutzutage sind fast alle Mikroprozessoren superskalar. Der Intel Pentium war Intels erster superskalarer Prozessor der IA32 der x86 ISA Familie. Auch andere Unternehmen wie AMD und Cyrix nutzten die IA32 und die 64-bit Erweiterung x86-64. Bis heute sind diese Architekturen zu sehr großen Teilen Rückwärtskompatibel. In dieser Sektion werden mit dem Intel Pentium (P5 Mikroarchitektur) und der Intel P6 Mikroarchitektur (Pentium II/Pro/III), sowie dem Cyrix 6x86 und den von AMD stammenden K5 und K6 Architekturen einige der wichtigsten Architekturen überblickt.

#### Intel Pentium.

Der Intel Pentium Prozessor basiert auf der P5 Mikroarchitektur. Eine detaillierte Beschreibung seiner Architektur gibt [1]. Der Prozessor ist deutlich performanter als sein Vorgänger i486, nicht nur aufgrund des neuen, superskalaren Designs. Trotzdem sind die einzelnen Integer-Pipelines ähnlich aufgebaut. Abbildung 12 zeigt die Pipeline Stages des Prozessors. Die U- und V-Integer-Pipelines bilden die Basis des Designs. An die U-Pipeline ist allerdings eine Fließkomma-Pipeline angehängt wodurch entsprechend Fließkommabefehle nur in dieser Pipeline bearbeitet werden können. Folgend werden die einzelnen Stages der Pipeline beschrieben.



**Figure 12: Intel Pentium Pipeline Stages (Bild: Modern Processor Design[36])**

Da x86 Architekturen einen CISC Befehlssatz besitzen, liegen Instruktionen in variabler Länge vor. Dadurch können sie oftmals nicht in einem Zyklus dekodiert werden. Durch Pipelining der „Decoding Stage“ kann dem entgegengewirkt werden.

Nachdem die Befehle aus dem Instruktionspuffer geladen und ausgerichtet wurden (aufgrund der variablen Befehlslänge) werden sie dekodiert. Verzweigungen werden bereits in der „Fetch Stage“ vorhergesagt. Hierfür wurde ein 256-Einträge breiter „Branch Target Buffer“ (BTB) verwendet.

In der „Decode1 Stage“ konnten simple Befehle direkt ausgeführt werden, während komplexere Instruktionen durch einen „Microcode Sequencer“ in Kontrollwörter umgewandelt werden mussten, um die Pipeline über mehrere Zyklen zu steuern. In der „Decode2 Stage“ wurden anschließend die Kontrollwörter dekodiert und an die „Execution Stage“ weitergeleitet. Außerdem wurden Adressen für den Zugriff auf den Speicher generiert.

Nachdem die Befehle an die „Execution Stage“ weitergeleitet wurden, haben die Funktionalen Einheiten Berechnungen durchgeführt oder es wurde auf den Daten-Cache zugegriffen. Bei Fließkommaberechnungen wurde der Befehl an die Fließkomma-Pipeline übergeben. Sind keine Fehler aufgetreten konnten die Operationen schließlich der „Writeback Stage“ übergeben werden. Dort wurden die Ergebnisse in die Register übertragen und damit der Zustand des Prozessors geändert.

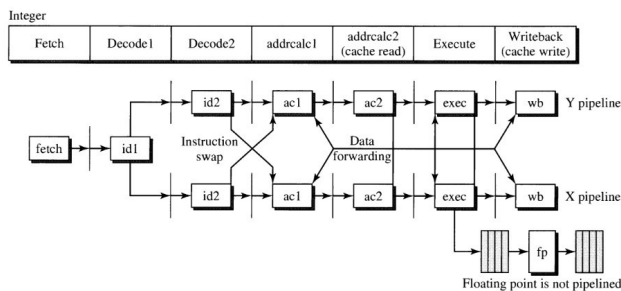
Der Pentium Prozessor konnte, mit einigen Einschränkungen, zwei Befehle gleichzeitig verarbeiten. Dazu wurde bei der Dekodierung festgestellt ob es sich um zwei simple Befehle handelt. War das der Fall, wurde zusätzlich überprüft, ob es sich bei der ersten Instruktion um einen Sprungbefehl handelt. War auch dies nicht der Fall, wurde überprüft ob



das Zielregister des ersten Befehls mit einem Register aus dem zweiten Befehl übereinstimmt. Bewahrheitete sich dies ebenfalls nicht, konnte der erste Befehl an die U-Pipeline und der zweite Befehl an die V-Pipeline übergeben werden. Ansonsten wurde nur die erste Instruktion in der U-Pipeline ausgeführt.

### Cyrix 6x86.

Der Cyrix6x86 hatte eine ähnliche Architektur wie der Intel Pentium Prozessor. Eine Ausarbeitung der Architektur gibt [22]. Weitere Informationen befinden sich im 6x86 Datenblatt[7]. Die beiden Integer-Pipelines wurden X und Y genannt, wobei die X-Pipeline eine Verknüpfung mit der Fließkomma-Pipeline besaß, ähnlich dem Design im Pentium. Weiterhin war die Fließkomma-Pipeline der Cyrix6x86 Architektur, im Gegensatz des Pentiums, nicht in mehrere Pipeline Phasen aufgeteilt. Abbildung 13 zeigt die Pipeline Phasen der Architektur.



**Figure 13: Cyrix 6x86 Pipeline Stages (Bild: Modern Processor Design[36])**

Im Gegensatz zum Intel Pentium hatte die Cyrix 6x86 Architektur insgesamt vier Phasen für die Befehlsdekodierung. In der „Decode1 Stage“ wurden die Länge der Instruktionen ausgewertet. Im nächsten Schritt wurden bis zu zwei Instruktionen zu den beiden Pipelines weitergeleitet. Dort wurden die Befehle dekodiert und zur „Address Calculation1 Stage“ übermittelt. Der Typ des Befehls wurde dabei berücksichtigt, um eine möglichst schnelle Verarbeitung zu ermöglichen. Anschließend wurden Speicheradressen berichtigt und darauffolgend in der „Address Calculation2 Stage“ etwaige Speicherverwaltungsaufgaben durchgeführt. Dazu gehörte der Zugriff auf den Cache und die Register. Zusätzlich wurden Fließkommabefehle an die Fließkommaeinheit übergeben. In der „Execution Stage“ wurden die Instruktionen ausgeführt, bevor die Ergebnisse in der „Writeback Stage“ in Register und Speicher zurückgeschrieben wurden. Eine weitere Abweichung des Pentiums war die Möglichkeit die Instruktionen in die jeweils andere Pipeline laden zu lassen, falls eine Pipeline die Verarbeitung der Befehle schneller abschloss als die andere Pipeline. Damit war es möglich, dass Instruktionen die „Writeback Stage“ nicht in sequentieller Reihenfolge erreichten. Entsprechend wurde überprüft, ob Abhängigkeiten zwischen Befehlen bestanden, sodass die entsprechende Pipeline verriegelt wurde bis diese aufgelöst wurden.

Es gab einige Instruktionen, die nur in der X-Pipeline ausgeführt werden konnten. Dazu gehörten exklusive Instruktionen, Befehle für Verzweigungen und Fließkommaberechnungen, wobei die letzten Beiden mit einer weiteren Instruk-

tion in der Y-Pipeline gepaart sein konnten.

Cyrix nutzte die Dual-Pipeline noch für weitere Designs. Zur Jahrtausendwende wurde das Unternehmen dann von Via aufgekauft und einige Jahre später die Entwicklung für weitere Cyrix Designs abgebrochen[36].

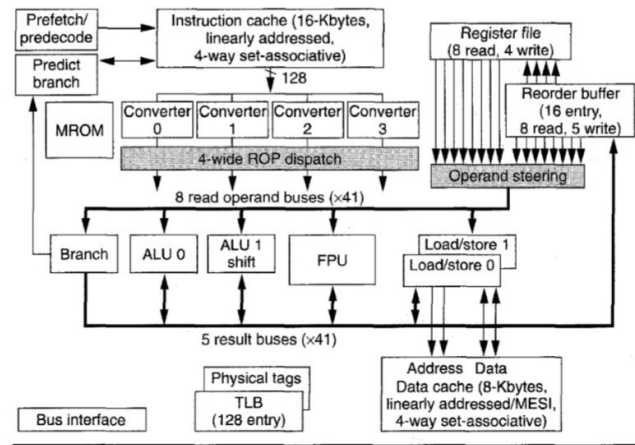
### Entkoppelte Architekturen.

Ende der 90er Jahre begannen Versuche entkoppelte Architekturen für die IA32 zu entwickeln. NexGen entwickelte den Nx586, der zwar nur eine Instruktion pro Zyklus verarbeiten konnte, intern aber superskalare Techniken umsetzte[36]. Angelehnt an die Arbeit von Yale Patt und seinen Studenten bezüglich limitierten Datenflusses[27], wurden die x86 Befehle in RISC86 Mikrooperationen übersetzt. Anschließend wurden diese dann an die entsprechenden Funktionalen Einheiten weitergeleitet.

Glenn Henry arbeitete beim Unternehmen Centaur ebenfalls an einer ähnlichen entkoppelten Lösung, der WinChip Serie. Das Unternehmen wurde allerdings mit Cyrix von Via aufgekauft[36].

### AMD K5.

AMD versuchte währenddessen eine Architektur zu entwerfen, die mit dem Intel Pentium konkurrieren konnte. Gleichzeitig wollte man eine eigene x86 Implementierung etablieren, auf der man aufbauen konnte. Die folgenden Ausführungen beziehen sich auf einen Artikel des IEEE Micro über die Entwicklung der K5 Architektur[5] von Dave Christie, damaligen Projektleiter des AMD K5.



**Figure 14: AMD K5 Blockdiagramm (Bild: Developing the AMD-K5 architecture[5])**

Die AMD K5 Architektur konnte maximal bis zu vier Befehle pro Zyklus verarbeiten. AMD hatte bereits einige Jahre vorher eine superskalare RISC Architektur (AM 29000) designt. Nachdem sich die Simulation dieser Architektur als sehr robust erwies, nutzte man das gesammelte Wissen beim Design einer neuen Mikroarchitektur. Ähnlich wie NexGen und Centaur setzte man auf eine Übersetzung der x86 zu RISC-ähnlichen Mikrooperationen, die AMD ROPs nannte.

Durch die komplexen x86 Befehle mit variabler Länge, hatte auch AMD Probleme die Instruktionen effizient zu dekodieren. Das Problem wurde gelöst, indem man Instruktionen beim Laden in den Instruktionscache bereits teilwei-

se dekodiert und insgesamt fünf Bits anhängte, die unter anderem Aufschluss über die Länge der Befehle gaben. Dadurch konnten dann die Dekodierer richtig ausgerichtet und die Befehle durch logisch simplere Dekodier-Einheiten verarbeitet werden.

Der „Execution Core“ bestand aus sechs Funktionalen Einheiten mit „Reservation Stations“. Die ROPs wurden ohne sequentielle Ordnung auf die Einheiten verteilt und konnten in zufälliger Ordnung abgearbeitet werden, solange ihre Operanden verfügbar und eine Funktionale Einheit bereit war. Der sequentielle Befehlsablauf wurde im „Reorder Buffer“ (ROB) verfolgt. Ebenfalls wurden hier spekulative Ergebnisse für Verzweigungen zwischengespeichert und „Register Renaming“ vorgenommen. War ein Befehl fertig abgearbeitet und alle vorherigen Befehle wurden bereits übernommen, konnte der echte Zustand des Prozessors geändert und das Ergebnis des Befehls in die Register übernommen werden.

Der K5 konnte über bis zu zwei nicht aufgelösten Verzweigungen hinaus Befehle verarbeiten. Im Gegensatz zu der damals weit verbreiteten Technik des „Branch Target Buffers“ wurden Vorhersagen mit Cache Lines (mit 16 Bytes Länge) verknüpft. Dadurch konnten mehr Vorhersagen mit nur einem geringen Speicherzuwachs realisiert werden.

### *AMD K6 und Nachfolger.*

Obwohl der K5 einige sehr fortschrittliche Techniken implementierte war er Leistungstechnisch nicht auf dem Stand der Konkurrenz. AMD übernahm 1995 NexGen und entwickelte den Nachfolger K6 hauptsächlich basierend auf dem dortigen, zu diesem Zeitpunkt entwickelten Nx686. Es wurden allerdings auch Techniken aus dem K5 implementiert, wie beispielsweise die Nutzung mehrerer Dekodierungseinheiten mit unterschiedlicher Komplexität oder die Nutzung von teilweiser Dekodierung im Instruktionscache.[36]

Nachfolger wie der AMD Athlon (K7) und der AMD Opteron (K8) basieren zu großen Teilen aus den Designs des K5 und K6. Allerdings schlug AMD die Erweiterung zur x86-64 ISA vor, welche von späteren Modellen unterstützt wurde.[36]

### *Intel P6 Core.*

Intel begann die Arbeit an der P6 Architektur 1990 und führte sie 1995 ein. Durch den Intel Pentium Pro Prozessor wurde die Architektur in Servern und Workstations eingesetzt. 1997 wurde der Pentium II veröffentlicht, welcher den Desktop Markt anvisierte. Der Pentium III wurde 1998 hinzugefügt. Durch Erweiterungen wurden Befehlssätze für MMX (Multimedia Extension) und SSE (Streaming SIMD Extensions) Befehle ergänzt. Damit konnten ebenfalls SIMD (Single Instruction Multiple Data) Befehle ausgeführt werden. Dabei handelt es sich um Befehle, die auf vielen Daten gleichzeitig ausgeführt werden, wie zum Beispiel Vektorberechnungen. J.P.Shen und M.H.Lipasti widmen dieser Architektur ein eigenes, sehr ausführliches Kapitel in [36]. In diesem Abschnitt wird ein Überblick über ihre Ausführungen geboten.

Die Architektur arbeitete mit 32-bit und war in der Lage drei Befehle gleichzeitig zu verarbeiten. Wie die zuvor besprochenen Architekturen nutzen die Prozessoren der P6 Generation „Out-of-Order Execution“ und spekulative Vorhersage von Verzweigungen. Ebenfalls wurden die x86 Befehle in Mikrooperationen übersetzt. Intel nennt diese Operatio-

nen  $\mu$ OPs. Die P6 Architektur fällt außerdem aufgrund der sehr fein gestaffelten Pipeline auf. Im Gegensatz zur Konkurrenz dieser Zeit, wurden die üblichen Pipeline Phasen in viele kleine Segmente aufgeteilt und über Warteschlangen verbunden. Dadurch erreichte die Architektur sehr viel höhere Taktraten. Das hatte den entscheidenden Nachteil, dass die Daten von großen Teilen der Pipeline gelöscht werden mussten, wenn die Vorhersage einer Verzweigung fehlerhaft war. Um den zu entgehen, nutzte Intel eine sehr akkurate Vorhersagetechnik und eine schnelle Anbindung zum Level 2 Cache.

In dieser Sektion wird die Architektur des P6 genauer unter die Lupe genommen. Im Gegensatz zu vorherigen Erläuterungen, geschieht dies in einem höheren Detailgrad, da abgesehen von Optimierungen und neuen Ansätzen zur Parallelisierung abseits von superskalaren Techniken, aktuelle Mikroprozessoren konzeptionell immer noch sehr ähnlich arbeiten. Die P6 Architektur gibt also einen guten Einblick in immer noch aktuelle Techniken bezüglich Superskalarität.

Die Architektur des P6 ist in drei Komponenten unterteilt. Im „Front End“ werden die Instruktionen aus dem Instruktionscache in den Instruktionspuffer geladen. Anschließend werden die Befehle dekodiert und an den „Reorder Buffer“ weitergeleitet. Dort warten sie auf nötige Operanden und werden bei Abwesenheit von Abhängigkeiten an den „Execution Core“ weitergeleitet. Ebenfalls findet dort die Vorhersage von etwaigen Verzweigungen mithilfe des „Branch Target Buffer“ statt.

Im „Execution Core“ werden die Instruktionen zuerst in eine zentrale „Reservation Station“ geladen. Dort werden sie bei Verfügbarkeit an die funktionalen Einheiten weitergeleitet, welche die nötigen Berechnungen durchführen. Speicherbefehle werden anschließend an den „Memory Reorder Buffer“ (MOB) übergeben. Die Ergebnisse der restlichen Befehle landen in den entsprechenden Einträgen des „Reorder Buffers“. In der „Retirement Stage“ warten sie nun darauf bis der „Reorder Buffer“ die Änderungen an die Register übergeben kann.

Abbildung 15 zeigt ein Blockdiagramm der P6 Pipeline. Im Folgenden werden die einzelnen Schritte erklärt, die Instruktionen in der entkoppelten Architektur durchlaufen.

„Front End“ Pipeline: Zuerst berechnet der „Branch Target Buffer“ eine Adresse für den Instruktionszeiger. Als nächstes wird die Cacheline an der berechneten Adresse aus dem Speicher geladen, um an die Dekodierungseinheiten gesendet zu werden. Bereits dieser Schritt ist in mehrere Segmente aufgeteilt, um möglichst viele Instruktionen in die Pipeline zu schleusen.

Anschließend werden die Daten aus der Cacheline dekodiert, um die Befehle mit variabler Länge zu isolieren. Danach können die Instruktionen in mehrere Mikrooperationen, den  $\mu$ OPs, übersetzt werden. Im nächsten Segment der Dekodierungsphase können Verzweigungen aufgespürt werden, die der „Branch Target Buffer“ möglicherweise nicht erkannt hat. Das passiert zum Beispiel, wenn eine Verzweigung das erste Mal ausgeführt wird. In der „Rename Stage“ werden die logischen Register der Operanden durch die „Register alias table“ (RAT) auf physische Register des „Reorder Buffers“ abgebildet (Register Renaming). Die Detektion von fehlerhaften Verzweigungen und Neubenennung der Operanden überschneiden sich zeitlich. Dies zeigt, wie einige Segmente der Pipeline des P6 miteinander verzahnt sind, um höhere Taktraten zu erreichen.

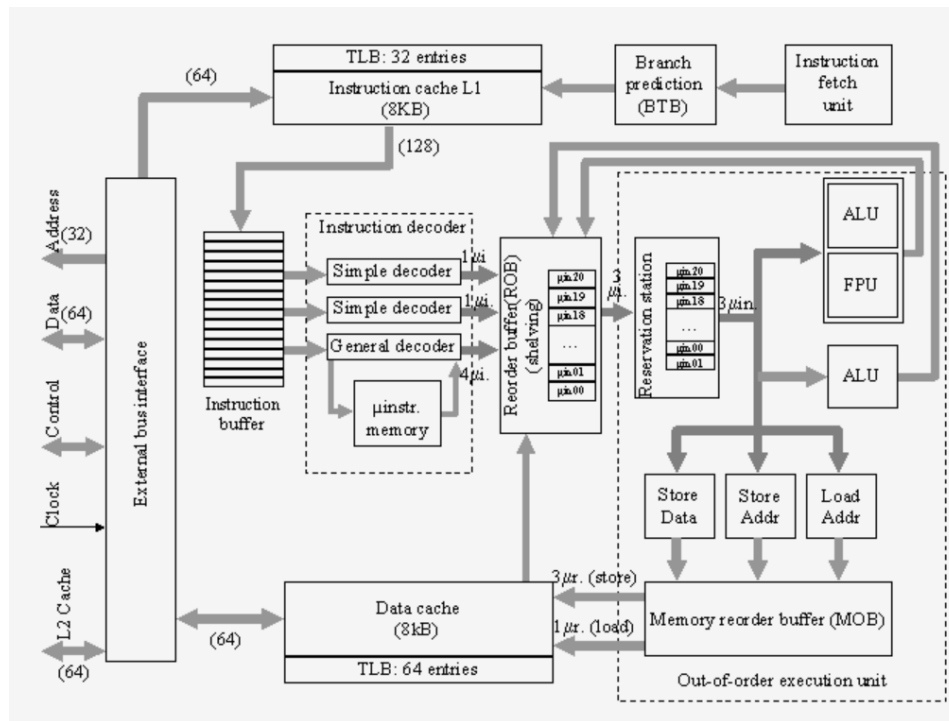


Figure 15: Intel P6 Blockdiagramm (Bild: Modern Processing Design[36])

Zuletzt werden bis zu drei  $\mu$ OPs gleichzeitig sowohl in den „Reorder Buffer“, als auch in die zentrale „Reservation Station“ des „Execution Cores“ geschrieben. Im „Reorder Buffer“ ist der Eintrag notwendig, um nach der Berechnung im „Execution Core“ den sequentiellen Programmablauf wiederherzustellen. Dies geschieht bereits hier, da ab der Übertragung in die „Reservation Station“ alle Befehle außerhalb der Programmordnung abgearbeitet werden können. In der „Reservation Station“ warten sie bis ihre Operanden, sowie eine Funktionale Einheit für den entsprechenden Befehlstyp verfügbar sind.

„Execution Core“: In der „Reservation Station“ müssen einige Bedingungen zutreffen, damit ein  $\mu$ OPs in eine Funktionale Einheit geladen wird:

- Alle Operanden sind verfügbar
- Eine Funktionale Einheit des passenden Typs muss verfügbar sein
- Ein Datenbus, um das Resultat nach der Berechnung weiterzuleiten, muss im entsprechenden Zyklus frei sein
- Die „Reservation Station“ muss dem in Frage kommenden  $\mu$ OPs die höchste Priorität, im Bezug, auf eine möglichst hohe Leistung der Maschine zuteilen

Entsprechend der oben gelisteten Bedingungen kann es dutzende oder gar hunderte Zyklen dauern bis ein  $\mu$ OP zugeteilt wird. Die „Reservation Station“ braucht sowohl einen Zyklus, um zu erkennen ob alle Operanden bereitstehen, als auch einen Zyklus, um den Befehl an die Funktionale Einheit zu übertragen.

Simple Operationen brauchen einen Zyklus für die Verarbeitung. Komplexere Operationen können einige Zyklen

in Anspruch nehmen. Hierzu gehören unter anderem Integer Multiplikationen oder Fließkommaberechnungen. Anschließend muss ein Datenbus frei sein, damit die Ergebnisse zurückgeschrieben werden können. Datenbusse werden zwischen den Funktionalen Einheiten geteilt, sodass die „Reservation Stations“ beachten müssen, wie viele Zyklen ein Befehl zur Ausführung benötigt.

Speicheroperationen nehmen eine besondere Stellung ein. Nachdem die Adresse generiert wurde, wird versucht den Operanden aus dem Datencache zu laden. Dafür werden zwei Zyklen benötigt. Verläuft alles positiv wird der Operand für andere  $\mu$ OPs zur Verfügung gestellt. Erfolgt ein Cache Miss wird zuerst der Level 2 Cache ausprobiert und bei erneutem Fehlschlag wird der Befehl zurückgestellt. In diesem Fall muss auf den Hauptspeicher zugegriffen werden, um die entsprechende Cacheline zu laden. Da dies sehr lange dauert, ist es effizienter die Ressourcen für andere Operationen freizusetzen bis die Daten geladen wurden. Dafür wird der „Memory Reorder Buffer“ (MOB) verwendet. Er verwaltet die wartenden Befehle und überprüft wann zurückgestellte Operationen aufgeweckt werden können. Daraufhin kann der Zugriff erneut erfolgen. Durch dieses Verfahren wird sichergestellt, dass verfügbare Ressourcen nicht leerlaufen während auf Daten gewartet wird.

„Retirement Pipeline“: Aufgrund der ungeordneten Ausführung des „Execution Cores“ muss die Programmführung wiederhergestellt werden, bevor der aktuelle Zustand der Maschine geändert wird. Hierzu übernimmt der „Reorder Buffer“ immer nur den ältesten Befehl. Da x86 Architekturen eine CISC Plattform darstellen, kann allerdings immer nur eine komplette x86 Instruktion übernommen werden. Aus diesem Grund sind die  $\mu$ OPs, die den anfang und das Ende der ursprünglichen Makrooperation darstellen, markiert.

So weiß der „Reorder Buffer“, dass er alle Mikrooperationen auf in einem Zyklus übernehmen kann. Der „Reorder Buffer“ ist zirkulär angeordnet. Neue Operationen überschreiben die Einträge hinter dem „Retirement Pointer“, während sich die ältesten Operationen unter beziehungsweise direkt vor dem Zeiger befinden.

Abschließend lässt sich sagen, dass die P6 Architektur von Intel sehr fortgeschrittene Techniken einsetzt. Sie entkoppelt die Pipeline in drei Teile, wobei „Front End“ und „Retirement Pipeline“ die Programmordnung respektieren. Gleichzeitig wird ermöglicht, dass der „Execution Core“ außerhalb dieser Ordnung arbeiten kann, um dessen Ressourcen bestmöglich zu nutzen.

Eine akkurate spekulative Ausführung mit vielen unabhängigen „Checkpoints“ trägt ebenfalls zur Leistungssteigerung bei. Wird eine falsche Vorhersage einer Verzweigung erkannt, kann das „Front End“ unabhängig sofort alle Operationen löschen und Instruktionen an der richtigen Adresse laden. Währenddessen ist der „Execution Core“ in der Lage alle Operationen bis zur Verzweigung weiterhin zu verarbeiten. Die „Retirement Pipeline“ ist indes in der Lage einen validen Zustand des Programms zu gewährleisten.

Die aktuellen Mikroarchitekturen Intels basieren bis heute auf dem Design der P6 Architektur und dessen Nachfolgern. Eine ausführliche Auseinandersetzung mit der Architektur findet sich in [16] und [36]. [4] und [25] setzen sich mit dem Pentium Pro Prozessor auseinander.

### Die Skylake Architektur - Ein aktueller Mikroprozessor.

In dieser Sektion soll ein aktueller Mikroprozessor betrachtet werden, um zu verdeutlichen wie die angesprochenen superskalaren Techniken auch heute noch Einzug in aktuelle Architekturen halten. Die Skylake Architektur wurde von Intel 2015 auf den Markt gebracht. Ihre Vorgänger begründen sich aus der Core-Technologie, die als Iteration der Architektur des P6 angesehen werden kann. Im Folgenden wird ausgeführt, wie viele Komponenten der P6 Architektur, auch 20 Jahre später immer noch vertreten sind. Wie die im letzten Paragraphen von Sektion 2.2.4 beschriebene P6 Architektur, besteht die Skylake Architektur aus einem „Front End“, einem „Execution Core“ und der „Retirement Pipeline“[28].

Abbildung 16 zeigt das „Front End“ der Skylake Architektur. Wie in der P6 Architektur werden Befehle anhand der Ergebnisse des „Branch Predictors“ aus dem Instruktionscache geladen. Ebenfalls gibt es einen Instruktionspuffer, welcher die dekodierten Instruktionen an die Dekodierungseinheiten weiterleitet. Wie schon im AMD K5 gesehen, werden Befehle teilweise vordekodiert, um sie den Dekodierungseinheiten besser zuordnen zu können. Durch die MacroOP Fusion können bestimmte Befehle gruppiert werden. Damit kann mit einem einzigen Befehl mehr umgesetzt werden und es werden zusätzliche Informationen gespart. Diese Befehle werden später zu gruppierten Mikrooperationen übersetzt. Dadurch wird Bandbreite gespart, mehr Befehle können am Ende gleichzeitig übernommen werden und die Größe des „Reorder Buffer“s fällt kleiner aus.

Nachdem die Befehle dekodiert wurden, warten sie in einem Puffer bis sie dem „Reorder Buffer“ zugeordnet werden. Währenddessen überprüft ein „Loop Stream Detector“ (LSD) ob Schleifen vorhanden sind, deren Befehle dann in den  $\mu$ OPs-Cache geladen werden. Somit müssen diese Ope-

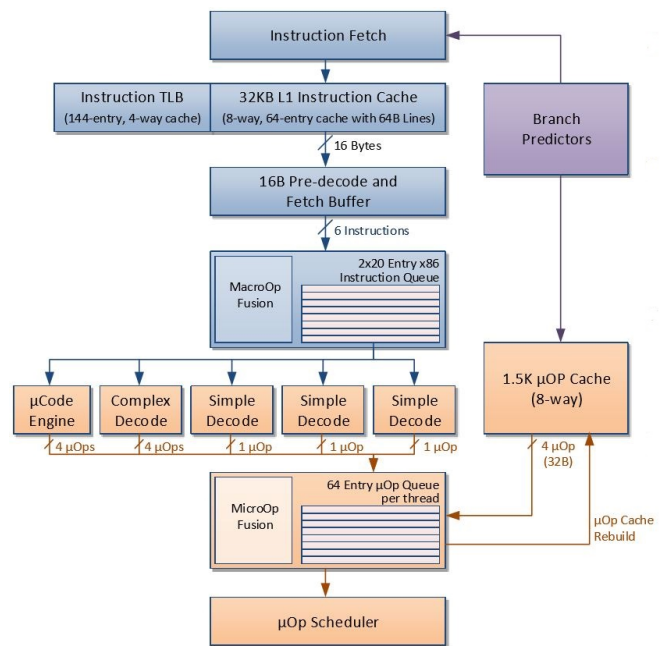


Figure 16: Skylake „Front End“ (Bild: Overview of Modern Microarchitectures[28])

rationen nicht mehrfach dekodiert werden, sondern können immer wieder aus dem Cache geladen werden.

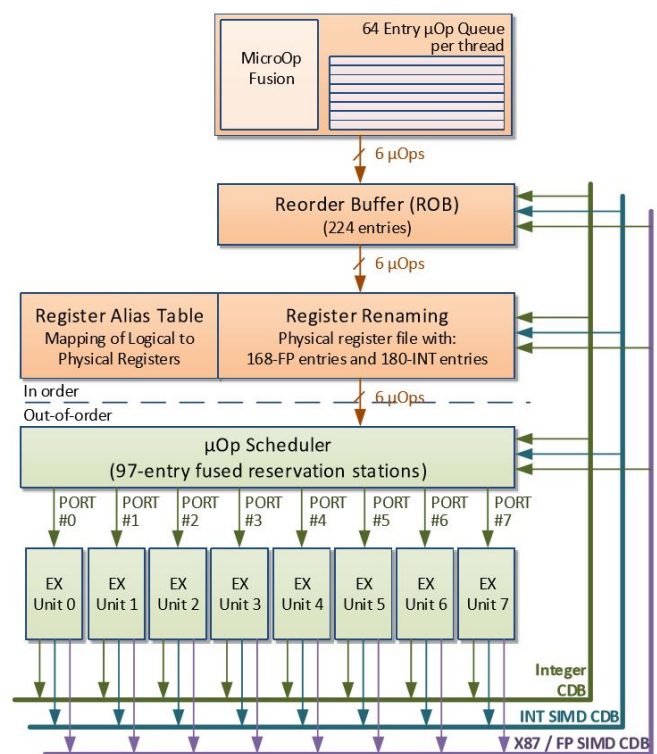


Figure 17: „Execution Core“ (Bild: Overview of Modern Microarchitectures[28])

Abbildung 17 zeigt den „Execution Core“ der Skylake Ar-



chitektur. Nachdem die Operationen in den „Reorder Buffer“ geladen wurden, werden die logischen Register der Operanden auf physische abgebildet. Anschließend können sie an die „Reservation Station“ ( $\mu$ OPs scheduler) weitergeleitet werden. Die funktionalen Einheiten berechnen die Ergebnisse und leiten sie über den „Common Data Bus“ weiter.

Diese Vorgehensweise ist sehr ähnlich zum Datenfluss in der P6 Architektur. Natürlich gibt es mehr funktionale Einheiten und die Puffer sind größer. Auch wurden einige Techniken weiter verbessert. Das Grundkonzept ist allerdings weiterhin das Gleiche. Sobald die Ergebnisse verfügbar sind und ältere Operationen übernommen worden sind, kann der „Reorder Buffer“ die Ergebnisse in die Register übernehmen.

Die alten Techniken bleiben also bestehen. Im Gegensatz zur P6 Architektur aus dem Jahre 1995 macht ein einzelner Kern natürlich nur noch einen geringen Teil der CPU aus. Neben mehreren Kernen und immer größeren Caches, besteht ein großer Teil der Fläche des Prozessors mittlerweile aus Komponenten für Vektorrechnung und Grafikeinheit.

## 2.3 Limits und neue Wege

In seiner Arbeit[23] von 1965 beschreibt Gordon Moore die Möglichkeiten für die, damals noch neue, Technologie der integrierten Schaltkreise. Er beschreibt, dass sich für mindestens zehn Jahre, also bis 1975, die Komplexität der Schaltkreise verdoppeln würde. Und selbst danach sei zu erwarten gewesen, dass sich der Trend ähnlich fortsetzen würde. Tatsächlich ist die Komplexität von Computerchips Jahrzehnte lang extrem schnell angestiegen und entsprechend auch die Geschwindigkeit von Prozessoren. Mittlerweile bestimmen nicht mehr Techniken wie Superpipelining und Superskalartät die großen Leistungszuwächse, sondern viel mehr die Anzahl der Prozessorkerne in einem Chip.

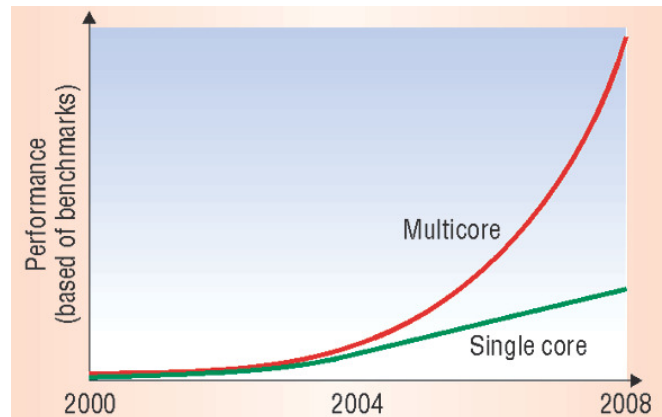
### 2.3.1 Leistungsaufnahme und Temperatur als limitierende Faktoren

2005 beschreibt David Geer die damals aktuellen Probleme bei der Herstellung immer performanteren Prozessoren[15]. Er stellt fest, dass in den neunziger Jahren die Leistung um 60% pro Jahr gestiegen ist, während dies in den zweitausender Jahren nur noch 40% betrug. Immer tiefere und breitere Pipelines haben also ihr Limit erreicht. Der Grund dafür sind laut Geer die immer kleiner werdenden Transistoren. Dadurch entsteht immer mehr Leistungsaufnahme und entsprechend Hitze die natürlich abgeleitet werden muss, um die Elektronik nicht zu beschädigen. Leistungsfähige Kühllösungen sind allerdings teuer. Außerdem hat komplexere Logik natürlich einen höheren Stromverbrauch. Da aber Leistungszuwachs und Komplexität nicht linear steigen, wird das irgendwann unrentabel. Das Verhältnis von zusätzlichem Platzverbrauch und Leistungssteigerung wird immer größer. Geer bezieht sich hierbei auf ein Zitat des damaligen Chefarchitekten von Sun, Marc Tremblay: "We could build a slightly faster chip, but it would cost twice the area while gaining only a 20 percent speed increase".

Er führt außerdem aus, woher die Probleme mit dem Stromverbrauch stammen. Er beschreibt das Problem wie folgt. Immer kleinere Gatter sind immer weniger in der Lage, den Elektronenfluss zu blockieren, wodurch auch außerhalb des Schaltens von Zuständen Strom verbraucht wird. Ebenfalls führen höhere Taktfrequenzen logischerweise zu häufigeren Zustandsänderungen, daraus resultiert ebenfalls mehr Stromverbrauch und Abwärme.

### 2.3.2 Der Weg zu Multicoreprozessoren

Um dem entgegen zu wirken werden einfach mehrere Kerne in einen Prozessor verfrachtet. Diese bringen zwar weniger Leistung mit als ein performanterer Einkernprozessor, haben allerdings auch einen kleineren Stromverbrauch und weniger Temperaturprobleme. Im Kollektiv können sie die Leistung des Prozessors, im Vergleich zum Einkernprozessor aber sehr deutlich steigern. Geer verweist hierbei auf die in Abbildung 18 zu sehende Grafik.



**Figure 18: Leistungsentwicklung nach Kernen (Bild: Computer[15])**

Weiterhin haben mehrere Kerne den Vorteil, das Arbeiten an mehreren Programmen auf die Kerne aufteilen zu können. Dadurch muss der Prozessor nicht so oft zwischen den unterschiedlichen Threads hin und her wechseln. Auch kann man einzelne Kerne abschalten oder alle Kerne auf einer niedrigeren Taktfrequenz arbeiten lassen.

Mittlerweile sind Mehrkernprozessoren der Standard. Intel bietet seit Ende 2018 den Xeon W-3185X an, der insgesamt 28 Kerne besitzt. Und auch im Smartphone-Markt sind im High-End Segment mittlerweile Prozessoren von bis zu acht Kernen im Umlauf.

### 2.3.3 Dark Silicon

Das Dennard Scaling (auch MOSFET Scaling genannt) beschreibt den Leistungsverbrauch von Transistoren relativ zu ihrer Größe. In seinem Papier[8] von 1974 hält Robert Dennard fest, dass durch die Verkürzung der Länge die Leistungsaufnahme von Transistoren auf gleicher Fläche gleichbleibend ist. Dementsprechend würden mehr, kleinere Transistoren nicht mehr Strom verbrauchen solange sich die Gesamtfläche nicht verändert. Wie oben beschrieben, hält diese Behauptung nicht mehr stand, da kleinere Transistoren den Elektronenfluss weniger blocken können. Dies hat Mitte der zweitausender Jahre zum Paradigmenwechsel von schnelleren Einkernprozessoren, hin zu den Mehrkernprozessoren geführt. Die ursprünglichen Techniken zur Parallelisierung durch mehr und weitere Pipelinestufen, wurde ersetzt durch die Vervielfältigung ganzer Prozessorkerne.

Allerdings geht das auch nicht zu beliebigen Faktoren. Durch zusätzliche Logik zur Verwaltung der Kerne wird ebenfalls zusätzliche Energie benötigt. Mit Prozessoren der 22nm Fertigungstechnik mussten bereits 21% des Prozessors abgeschaltet sein, um keine Probleme mit der Leistungsaufnahme, beziehungsweise Hitzeentwicklung zu bekommen. Man

nutzt die hohe Unwahrscheinlichkeit aus, dass der komplette Prozessor (also auch jeder Kern) ausgelastet ist. Diese pausierten, abgeschalteten Bereiche werden als Dark Silicon bezeichnet. Für die 8nm Fertigungstechnik wird erwartet, dass über 50% des Chips pausiert werden müssen. [10]

### 2.3.4 Ausblick zur Entwicklung zukünftiger Mikroprozessoren

Durch die Restriktionen der Leistungsaufnahme und Temperaturentwicklung erreichen heutige Techniken wohl bald ihr Limit[9]. Nachdem die Mehrkernprozessoren eingeführt wurden, um dies zu umgehen steht man auch mit dieser Technik seit einiger Zeit vor diesem Problem. Es scheint, als würde es immer wichtiger werden, möglichst effiziente Kühlösungen zu entwickeln und die Leistungsaufnahme zu verringern. Nur so kann gewährleistet werden, dass weiterhin große Leistungssteigerungen möglich sind.

Die Abschaltung einiger Komponenten des Prozessors helfen hier bereits. Auch spezialisierte Prozessoren wie Grafikkarten und Vektorprozessoren bieten Möglichkeiten große Leistung mit möglichst geringem Stromverbrauch zu gewährleisten.

## 2.4 Fazit

Parallelität auf Befehlsebene und insbesondere Superskalarität sind einige der größten Entwicklungen in der Rechnerarchitektur und sind heutzutage in fast jedem Prozessor vorhanden, es sei denn, es gelten strenge Leistungs- oder Kostenbeschränkungen. Diese Techniken haben die Leistung auf eine Art und Weise verbessert, die vor den sechziger Jahren kaum vorstellbar war und den IPC-Wert weit über eins ansteigen ließen, einen Wert, den frühere Computer kaum erreicht haben.

In dieser Seminararbeit haben wir diesen Fortschritt in seinen historischen Kontext gestellt und seine Entwicklung seit den Anfängen mit den Computern CDC 6600 und IBM System/360 Model 91 erörtert. Damals war der wichtigste architektonische Fortschritt das Hinzufügen von mehreren (gleichen) Funktionseinheiten. Auf diese Weise konnte mehr als ein Befehl, auch vom selben Typ, gleichzeitig ausgeführt werden. Außerdem können gestoppte Zyklen aufgrund blockierter funktionaler Einheiten oder fehlender Operanden, dank „OoO-Execution“ besser ausgenutzt werden, während Abhängigkeiten im Programmcode respektiert werden.

Aufgrund dieser Möglichkeiten erforschte man in den siebziger und achtziger Jahren verschiedene Technologien, um den IPC-Wert über eins anwachsen zu lassen. Neben Vektorprozessoren, VLIW-Architekturen und Multiprozessoren, wurden ab Ende der achtziger Jahre vor allem superskalare Rechner entwickelt. Wie sehr die Ideen der letzten fünfzig Jahre diese Technologien geprägt haben, ist eindrucksvoll an der Architektur heutiger Mikroprozessoren zu erkennen.

Trotz aller Erfolge, konnte durch Limitierungen wie exponentiell steigender Stromverbrauch und Hitzeabstrahlung letztendlich immer weniger Leistungsgewinn verbucht werden. Mehrkernprozessoren konnten zeitweise zwar Abhilfe schaffen, doch stehen Computerarchitekten heute auch bei dieser Technologie vor demselben Problem. Es bleibt abzuwarten, wie die verschiedenen Hersteller sich dieser Herausforderung stellen und welche innovativen Technologien daraus entstehen werden. Eines ist aber sicher, eine spannende und aufregende Zeit in der Rechnerarchitektur steht uns bevor.

## 3. ACKNOWLEDGMENTS

Wir möchten uns bei unserem Betreuer Amir Raoofy für die Zeit bedanken, die er aufgewendet hat, um diese Arbeit mit uns zu strukturieren und zu korrigieren. Wir bedanken uns auch bei unseren Veranstaltern, Dr.-Ing. Trinitis und Dai Yang, die uns die Möglichkeit gegeben haben, an diesem Seminar im renommierten Deutschen Museum teilzunehmen. Zu erfahren, wie frühere Computer entstanden sind, war eine bereichernde Erfahrung, die uns geholfen hat, unser Wissen auf dem Gebiet der Rechnerarchitektur zu festigen und die vergangenen, gegenwärtigen und zukünftigen Herausforderungen besser zu verstehen.

## 4. REFERENCES

- [1] D. Alpert and D. Avnon. Architecture of the pentium microprocessor. *IEEE Micro*, 13(3):11–21, June 1993.
- [2] D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo. The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling. *IBM Journal of Research and Development*, 11(1):8–24, jan 1967.
- [3] A. D. Berenbaum, B. W. Colbry, D. R. Ditzel, R. D. Freeman, H. R. McLellan, K. J. O'Connor, and M. Shoji. Crisp: a pipelined 32-bit microprocessor with 13-kbit of cache memory. *IEEE Journal of Solid-State Circuits*, 22(5):776–782, Oct 1987.
- [4] D. Bhandarkar and J. Ding. Performance characterization of the pentium pro processor. In *Proceedings Third International Symposium on High-Performance Computer Architecture*, pages 288–297, Feb 1997.
- [5] D. Christie. Developing the amd-k5 architecture. *IEEE Micro*, 16(2):16–27, April 1996.
- [6] H. G. Cragon and W. J. Watson. The ti advanced scientific computer. *Computer*, 22(1):55–64, Jan 1989.
- [7] Cyrix. *6x86 Processor: Superscalar, Superpipelined, Sixth-generation, x86 Compatible CPU*.
- [8] R. H. Dennard, F. H. Gaensslen, Hwa-Nien Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc. Design of ion-implanted mosfet's with very small physical dimensions. *Proceedings of the IEEE*, 87(4):668–678, April 1999.
- [9] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, June 2011.
- [10] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Power limitations and dark silicon challenge the future of multicore. *ACM Trans. Comput. Syst.*, 30(3):11:1–11:27, Aug. 2012.
- [11] J. A. Fisher. Very long instruction word architectures and the eli-512. *IEEE Solid-State Circuits Magazine*, 1(2):23–33, Spring 2009.
- [12] M. Flynn. Computer engineering 30 years after the IBM Model 91. *Computer*, 31(4):27–31, apr 1998.
- [13] M. J. Flynn and P. R. Low. The IBM System/360 Model 91: Some Remarks on System Development. *IBM Journal of Research and Development*, 11(1):2–7, jan 1967.
- [14] C. C. Foster and E. M. Riseman. Percolation of code to enhance parallel dispatching and execution. *IEEE*

- Transactions on Computers*, C-21(12):1411–1415, Dec 1972.
- [15] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, May 2005.
  - [16] L. Gwennap. Intel’s p6 uses decoupled superscalar design. *Microprocessor Report*, 9(2):9–15, 1995.
  - [17] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. 2011.
  - [18] C. Kozyrakis and D. Patterson. Vector vs. superscalar and vliw architectures for embedded multimedia benchmarks. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 35, pages 283–293, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.
  - [19] L. Kurian, P. T. Hulina, and L. D. Coraor. Memory latency effects in decoupled architectures. *IEEE Transactions on Computers*, 43(10):1129–1139, Oct 1994.
  - [20] D. May, R. Shepherd, and P. Thompson. The t9000 transputer. In *Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers Processors*, pages 209–212, Oct 1992.
  - [21] S. McGeedy. The i960ca superscalar implementation of the 80960 architecture. In *Digest of Papers Compcon Spring ’90. Thirty-Fifth IEEE Computer Society International Conference on Intellectual Leverage*, pages 232–240, Feb 1990.
  - [22] S. C. McMahan, M. Bluhm, and R. A. Garibay. 6/spl times/86: the cyrix solution to executing /spl times/86 binaries on a high performance microprocessor. *Proceedings of the IEEE*, 83(12):1664–1672, Dec 1995.
  - [23] G. E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, Jan 1998.
  - [24] O. Mutlu. Computer Architecture Lecture 10 : Out-of-Order Execution Review: Solutions to Enable Precise Exceptions. 2011.
  - [25] D. B. Papworth. Tuning the pentium pro microarchitecture. *IEEE Micro*, 16(2):8–15, April 1996.
  - [26] R. Patel and K. Yarlagadda. Testability features of the supersparc microprocessor. In *Proceedings of IEEE International Test Conference - (ITC)*, pages 773–781, Oct 1993.
  - [27] Y. N. Patt, S. W. Melvin, W. M. Hwu, M. C. Shebanow, and C. Chen. Run-time generation of hps microinstructions from a vax instruction stream. *SIGMICRO Newsl.*, 17(4):75–81, Dec. 1986.
  - [28] T. Pedro. Overview of modern Micro-Architectures. 2013.
  - [29] R. N. L. Pugh, W. IBM’s 360 and early 370 systems, 1991.
  - [30] C. J. Purcell. The control data star-100: Performance measurements. In *Proceedings of the May 6-10, 1974, National Computer Conference and Exposition*, AFIPS ’74, pages 385–387, New York, NY, USA, 1974. ACM.
  - [31] B. R. Rau and J. A. Fisher. Instruction-level parallel processing: History, overview, and perspective. *The Journal of Supercomputing*, 7(1-2):9–50, may 1993.
  - [32] B. R. Rau, C. D. Glaeser, and R. L. Picard. Efficient code generation for horizontal architectures: Compiler techniques and architectural support. In *Proceedings of the 9th Annual Symposium on Computer Architecture*, ISCA ’82, pages 131–139, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.
  - [33] R. M. Russell. The cray-1 computer system. *Commun. ACM*, 21(1):63–72, Jan. 1978.
  - [34] K. Sankaralingam, R. Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, D. Burger, S. W. Keckler, and C. R. Moore. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. In *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*, pages 422–433, June 2003.
  - [35] H. Sharangpani and H. Arora. Itanium processor microarchitecture. *IEEE Micro*, 20(5):24–43, Sep. 2000.
  - [36] J. P. Shen and M. H. Lipasti. *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.
  - [37] Smith, Weiss, and Pang. A simulation study of decoupled architecture computers. *IEEE Transactions on Computers*, C-35(8):692–702, Aug 1986.
  - [38] J. E. Smith. Decoupled access/execute computer architectures. *SIGARCH Comput. Archit. News*, 10(3):112–119, Apr. 1982.
  - [39] J. E. Smith. Dynamic instruction scheduling and the astronautics zs-1. *Computer*, 22(7):21–35, July 1989.
  - [40] J. E. Smith, G. E. Dermer, B. D. Vanderwarn, S. D. Klinger, C. M. Rozewski, D. L. Fowler, K. R. Scidmore, and J. P. Laudon. The astronautics zs-1 processor. In *Proceedings 1988 IEEE International Conference on Computer Design: VLSI*, pages 307–310, Oct 1988.
  - [41] J. E. Smith and G. S. Sohi. The Microarchitecture of Superscalar Processors. *Proceedings of the IEEE*, 83(12):1609–1624, 1995.
  - [42] M. K. Smotherman, E. H. Sussenguth, and R. J. Robelen. The ibm acs project. *IEEE Annals of the History of Computing*, 38(1):60–74, Jan 2016.
  - [43] J. E. Thornton. *Design of a Computer - The Control Data 6600*.
  - [44] J. E. Thornton. The CDC 6600 Project. *IEEE Annals of the History of Computing*, 2(4):338–348, oct 1980.
  - [45] G. S. Tjaden and M. J. Flynn. Detection and parallel execution of independent instructions. *IEEE Transactions on Computers*, C-19(10):889–895, Oct 1970.
  - [46] R. M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *Cycle*, 34(January):25–33, 1967.
  - [47] T. J. Watson. Memorandum Regarding the CDC 6600 (August 28, 1963).
  - [48] M. Wilkes and J. Stringer. Micro-programming and the Design of the Control Circuits in an Electronic Digital Computer, 1952.