Vorlesung: Rechnerstrukturen, Teil 2 (Modul IP7)

J. Zhang

zhang@informatik.uni-hamburg.de

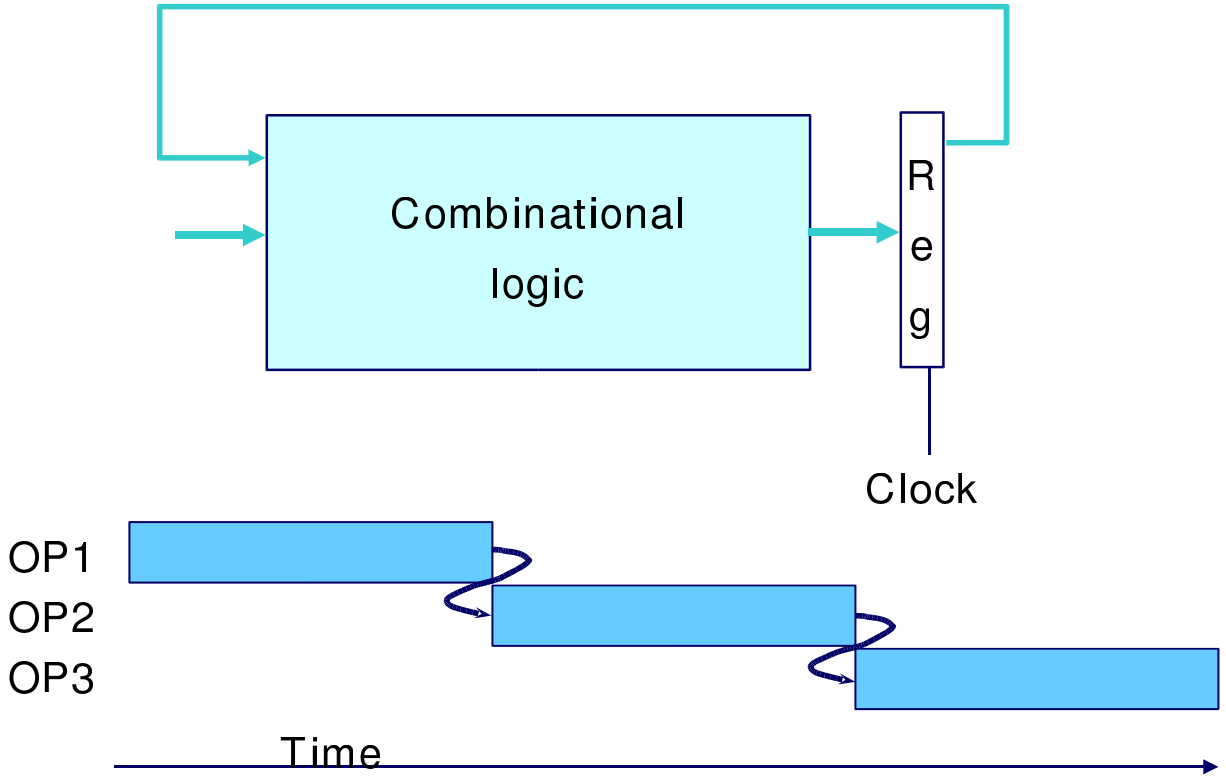
Universität Hamburg

Fachbereich Informatik

AB Technische Aspekte Multimodaler Systeme

Inhaltsverzeichnis

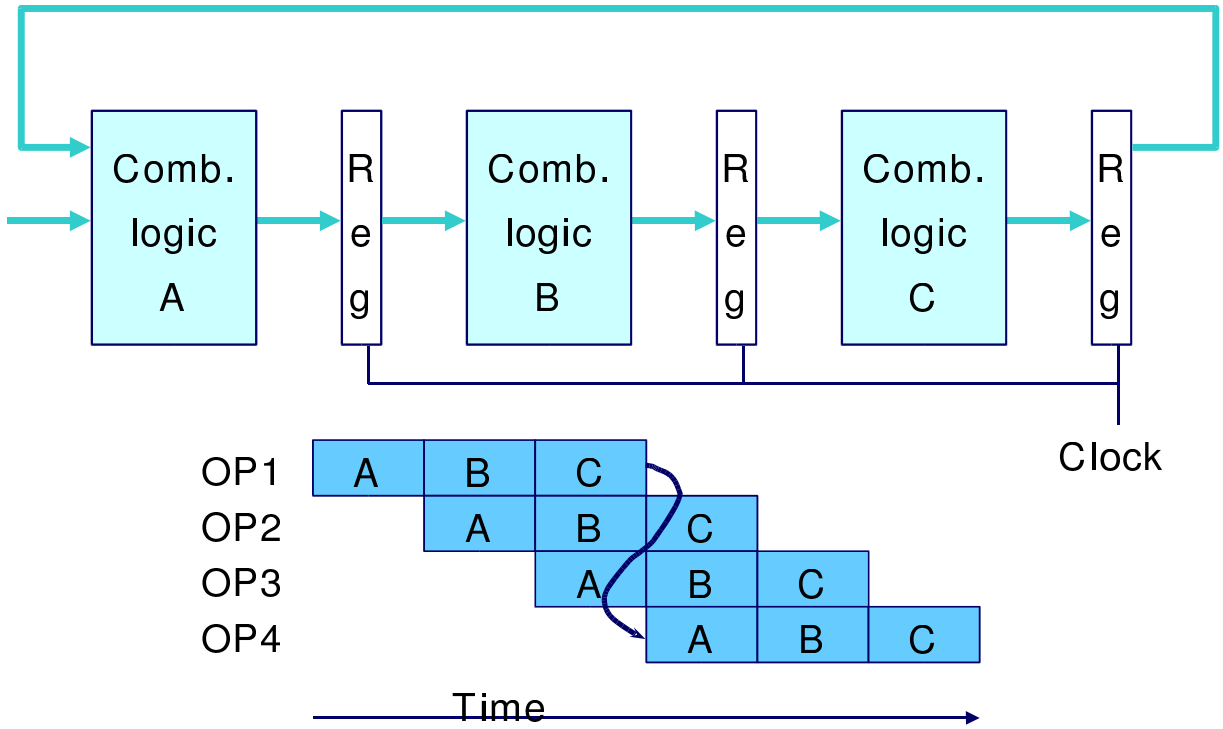
Datenabhängigkeiten



System

- Jede Operation hängt vom Erebnis der vorhergehenden ab

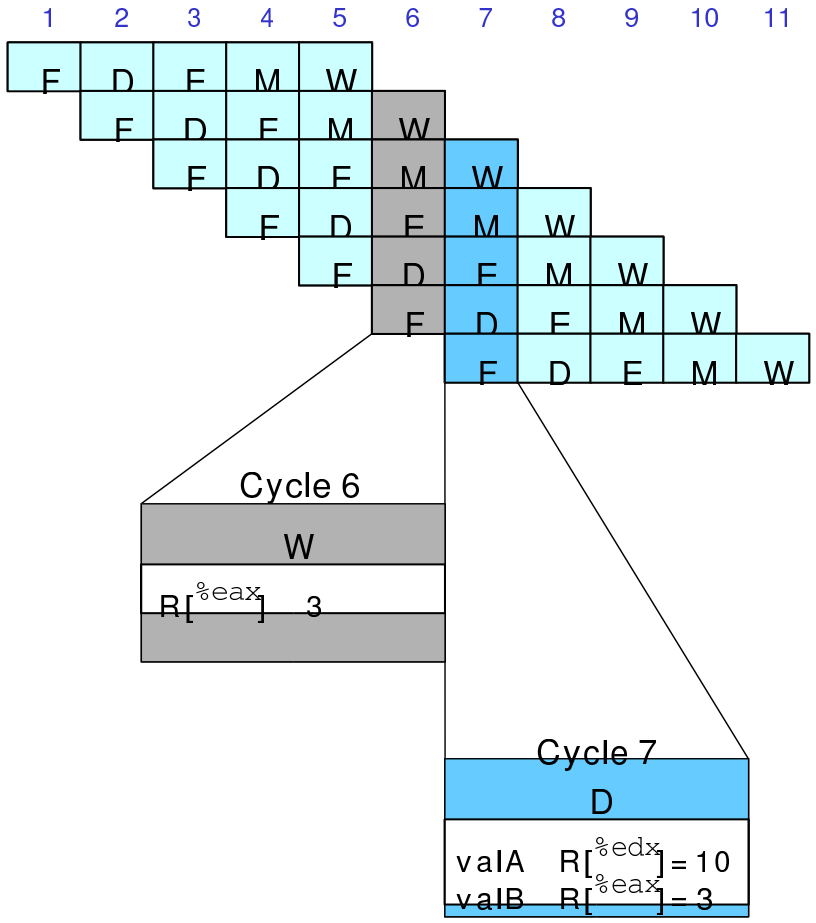
Daten Hazards



- Resultat-Feedback kommt zu spät für die nächste Operation
- Pipelining ändert Verhalten des gesamten Systems

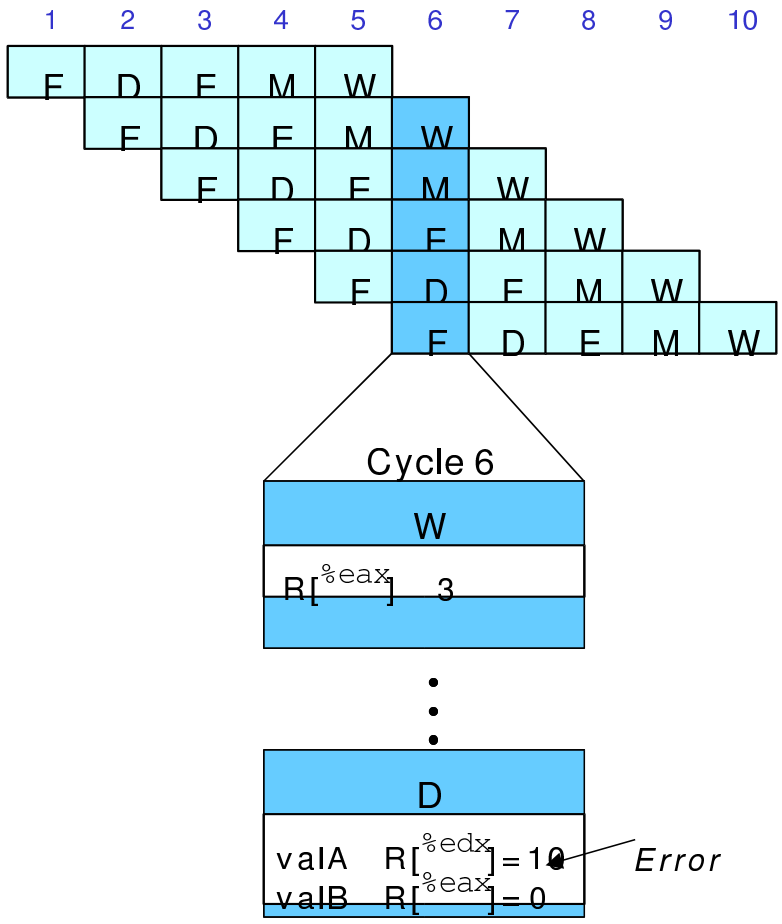
Datenabhängigkeiten: 3 Nops (No Operation)

```
# demo-h3.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
0x00e: nop
0x00f: addl %edx,%eax
0x011: halt
```



Datenabhängigkeiten: 2 Nops

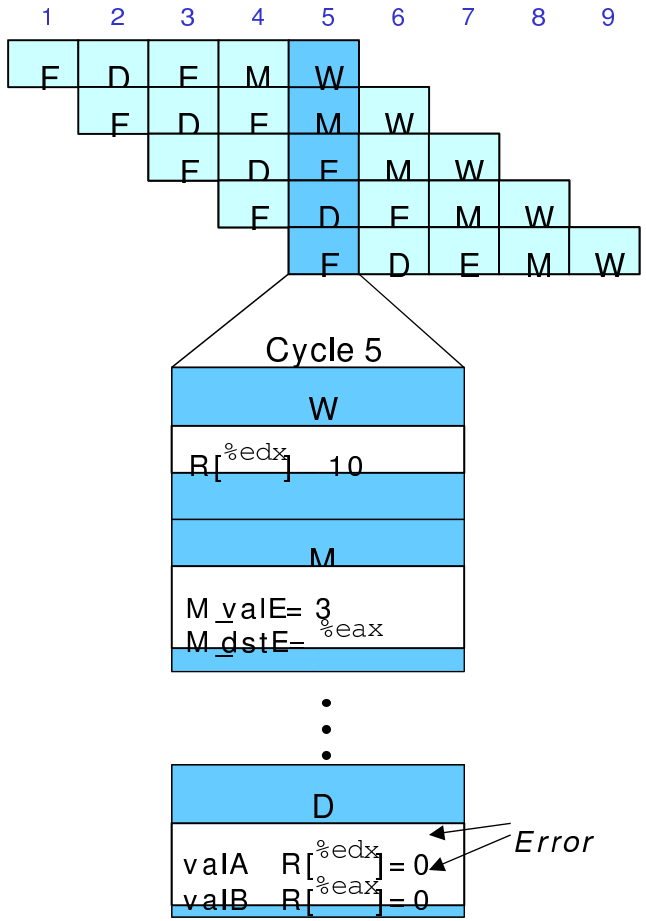
```
# demo-h2.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
0x00e: addl %edx,%eax
0x010: halt
```



Datenabhängigkeiten: 1 Nops

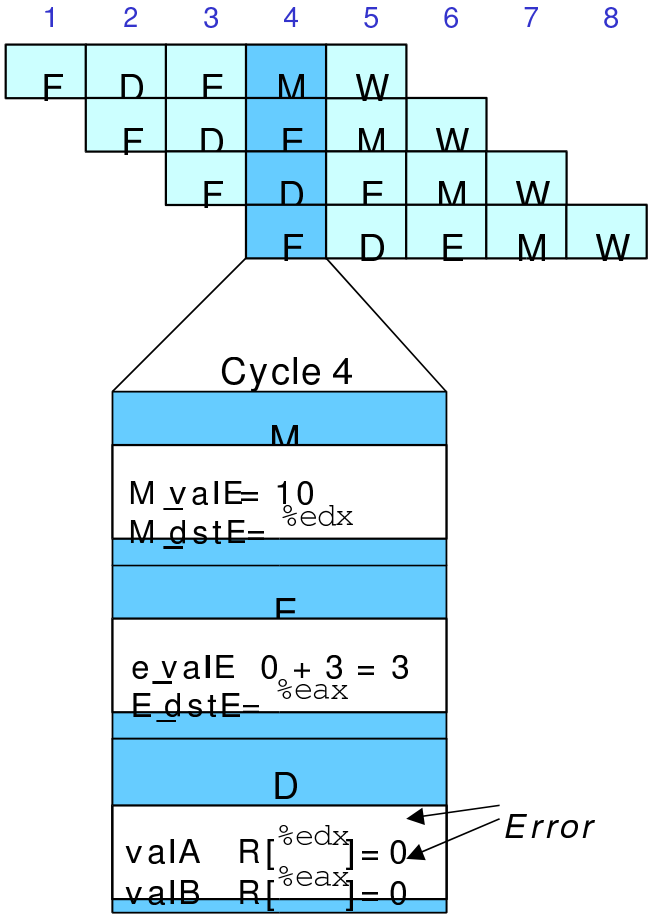
```
# demo-h1.ys

0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: addl %edx,%eax
0x00f: halt
```



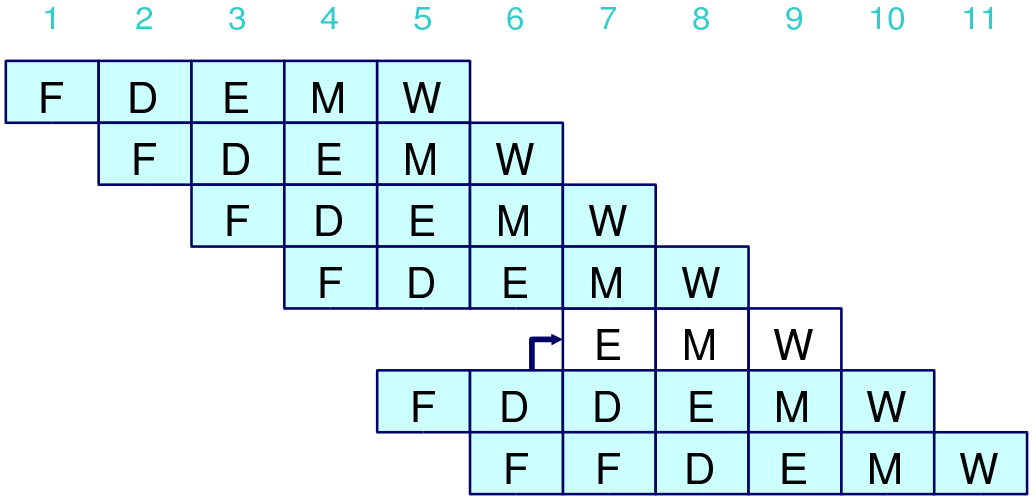
Datenabhängigkeiten: No Nop

```
# demo-h0.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: addl %edx,%eax
0x00e: halt
```



"Stalling" für Datenabhängigkeiten

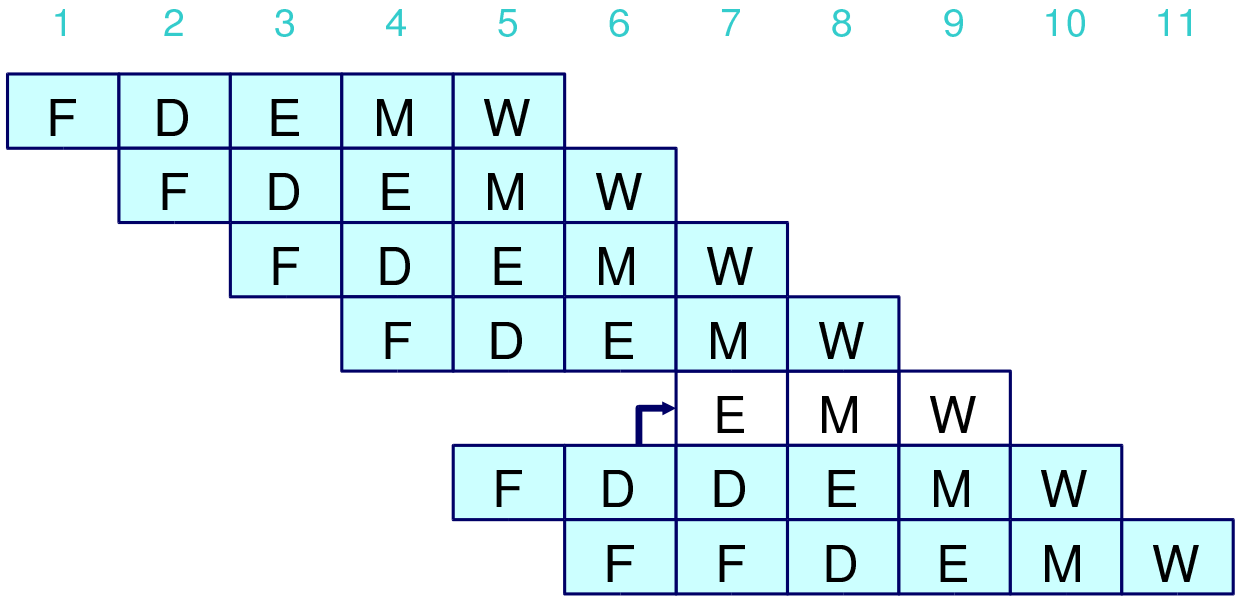
```
# demo-h2.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
       bubble
0x00e: addl %edx,%eax
0x010: halt
```



- Wenn die Anweisung zu dicht auf eine andere folgt, die in das von der aktuellen Anweisung benutzte Register schreibt, dann verzögere Anweisung
- Halte Anweisung im Decode-Status
- Füge nop dynamisch in "Execute Stage" ein

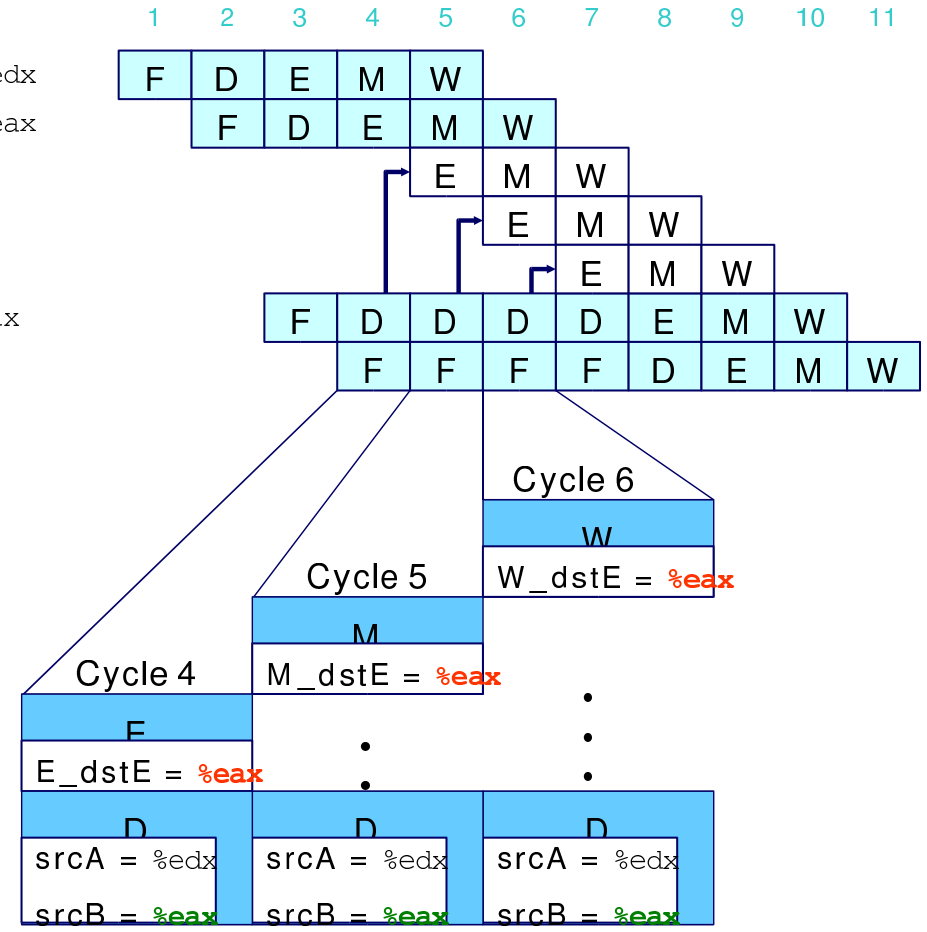
Feststellen der Stall Bedingung

```
# demo-h2.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
       bubble
0x00e: addl %edx,%eax
0x010: halt
```



Stalling X3

```
# demo-h0.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
       bubble
       bubble
       bubble
0x00c: addl %edx,%eax
0x00e: halt
```



Was passiert beim Stalling?

```
# demo-h0.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: addl %edx,%eax
0x00e: halt
```

Cycle 8

Write Back	***bubble***
Memory	***bubble***
Execute	`0x00c: addl %edx,%eax`
Decode	`0x00e: halt`
Fetch	

- Verzögerte Instruktion wird im "Decode"-Status gehalten
- Folgende Anweisung verbleibt im "Fetch" Status (Befehl-holen-Status)
- Bubbles werden in "Execute Stage" eingefügt
  - ✦ Wie dynamisch generierte nops
  - ✦ Bewegen sich durch spätere Stufen (stages)

"Data Forwarding"

Naive Pipeline

- Register wird erst nach Vollendung der "Write-back" Phase (stage) geschrieben
- Source-Operanden werden aus der Registerbank während der Dekodierungsphase gelesen
  - ✦ Müssen zu Beginn der Phase in der Registerbank sein

"Data Forwarding"

Naive Pipeline

- Register wird erst nach Vollendung der "Write-back" Phase (stage) geschrieben
- Source-Operanden werden aus der Registerbank während der Dekodierungsphase gelesen
  - ✦ Müssen zu Beginn der Phase in der Registerbank sein

Beobachtung

- Ergebnis wird während der "Execute Stage" oder der "Memory Stage" generiert

"Data Forwarding"

Naive Pipeline

- Register wird erst nach Vollendung der "Write-back" Phase (stage) geschrieben
- Source-Operanden werden aus der Registerbank während der Dekodierungsphase gelesen
  - ✦ Müssen zu Beginn der Phase in der Registerbank sein

Beobachtung

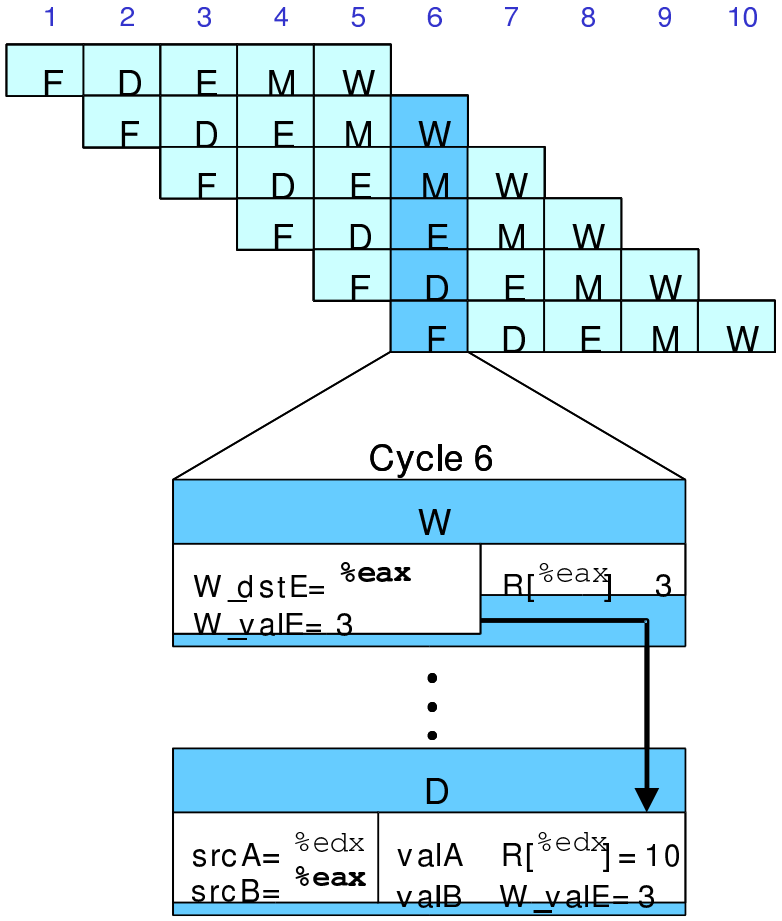
- Ergebnis wird während der "Execute Stage" oder der "Memory Stage" generiert

Trick

- Leite den Wert direkt von der generierenden Anweisung in die Dekodierungsphase
- Muss zum Ende der Dekodierungsphase verfügbar sein

Beispiel für "Data Forwarding"

```
# demo-h2.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
0x00e: addl %edx,%eax
0x010: halt
```



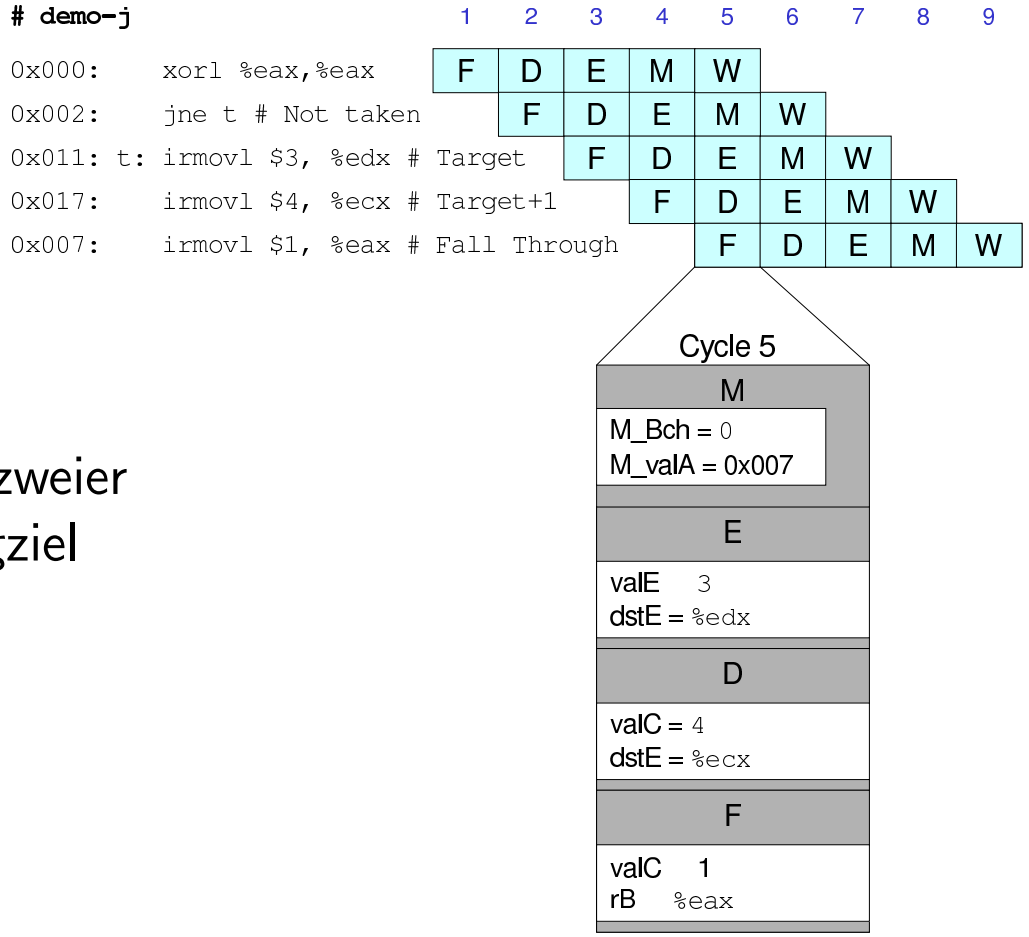
- irmovl in der "Write-back" Phase
- Zielwert im W Pipeline Register
- Weiterleiten als valB in die Dekodierungsphase

Beispiel für falsche Sprungvorhersage

```
0x000:     xorl %eax,%eax
0x002:     jne  t                # Not taken
0x007:     irmovl $1, %eax       # Fall through
0x00d:     nop
0x00e:     nop
0x00f:     nop
0x010:     halt
0x011: t:  irmovl $3, %edx       # Target (Should not execute)
0x017:     irmovl $4, %ecx       # Should not execute
0x01d:     irmovl $5, %edx       # Should not execute
```

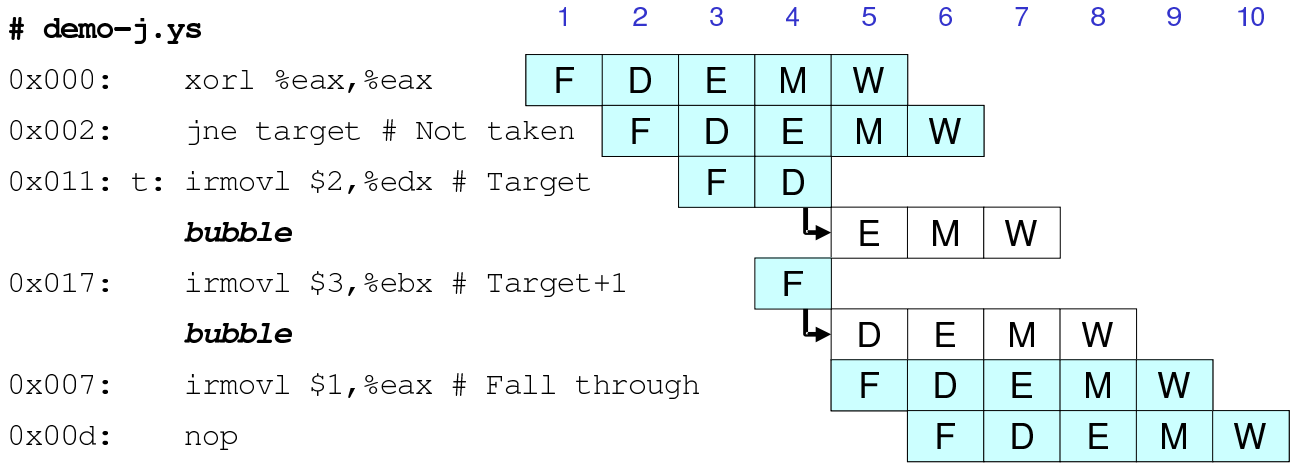
- Sollte nur die ersten 8 Anweisungen ausführen

Trace einer falschen Sprungvorhersage



- Fälschliche Ausführung zweier Anweisungen am Sprungziel

Behandlung einer falschen Sprungvorhersage (Branch Misprediction)



Sprung wird vorhergesagt

- Holt die nächsten 2 Anweisungen am Sprungziel

Abbruch bei falscher Vorhersage

- Erkenne den nicht ausgeführten Sprung in der “Execute”-Phase
- Im folgenden Zyklus, ersetze die in der “Execute”-Phase befindliche Anweisung und fülle Dekodierungsphase mit Bubbles auf
- Es sind noch keine Seiteneffekte aufgetreten

Pipeline Zusammenfassung

Data Hazards

- Meist durch Forwarding umgangen
  - ✦ Kein Performance Verlust
- Load/Use Hazards erfordern Verzögerung um einen Takt

Pipeline Zusammenfassung

Data Hazards

- Meist durch Forwarding umgangen
  - ✦ Kein Performance Verlust
- Load/Use Hazards erfordern Verzögerung um einen Takt

Control Hazards

- Breche Anweisungen ab, wenn ein falsch vorghergesagter Sprung entdeckt wird
  - ✦ Zwei Taktzyklen verschwendet
- Verzögere "Fetch Stage" während ret durch die Pipeline wandert
  - ✦ Drei Taktzyklen verschwendet

Pipeline Zusammenfassung II

Control Kombinationen

- Müssen sorgfältig analysiert werden
- Erste Version hat einen schwer erkennbaren Bug
  - ✦ Trat nur bei unüblicher Kombination von Befehlen auf

Ausnahmen

- Bedingungen, unter denen die “Pipeline” den normalen Ablauf nicht fortsetzen kann

Ausnahmen

- Bedingungen, unter denen die “Pipeline” den normalen Ablauf nicht fortsetzen kann

Ursachen

- Halt Anweisung
- Ungültige Adresse für die Anweisung oder Daten
- Ungültige Anweisung
- “Pipeline” Kontrollfehler

Ausnahmen

- Bedingungen, unter denen die "Pipeline" den normalen Ablauf nicht fortsetzen kann

Ursachen

- Halt Anweisung
- Ungültige Adresse für die Anweisung oder Daten
- Ungültige Anweisung
- "Pipeline" Kontrollfehler

Erforderliches Vorgehen

- Einige Anweisungen vollenden
  - ✦ Entweder aktuelle oder vorherige (hängt von Ausnahmetyp ab)
- Andere verwerfen
- "Exception handler" aufrufen
  - ✦ Wie unerwarteter Prozeduraufruf

PentiumPro Operation

Übersetzt Anweisungen dynamisch in "μops"

- 118 Bits breit
- Enthält Operation, zwei Quellen und Ziel

PentiumPro Operation

Übersetzt Anweisungen dynamisch in "μops"

- 118 Bits breit
- Enthält Operation, zwei Quellen und Ziel

Führt μops mit "Out of Order" Maschine aus

- μop ausgeführt wenn
  - Operanden verfügbar sind
  - Funktionelle Einheit verfügbar ist
- Ausführung wird durch "Reservation Stations" kontrolliert
  - Beobachtet die Datenabhängigkeiten zwischen μops
  - Teilt Ressourcen zu

Pentium 4 Eigenschaften

"Trace" Cache

- Ersetzt die traditionelle Anweisungscache
- Nimmt Anweisungen in dekodierter Form in die Cache auf
- Reduziert die benötigte Rate für den Anweisungsdecoder

Pentium 4 Eigenschaften

"Trace" Cache

- Ersetzt die traditionelle Anweisungscache
- Nimmt Anweisungen in dekodierter Form in die Cache auf
- Reduziert die benötigte Rate für den Anweisungsdecoder

"Double pumped" ALUs (2 Operationen pro Taktzyklus)

- Simple Anweisungen (hinzufügen)

Pentium 4 Eigenschaften

"Trace" Cache

- Ersetzt die traditionelle Anweisungscache
- Nimmt Anweisungen in dekodierter Form in die Cache auf
- Reduziert die benötigte Rate für den Anweisungsdecoder

"Double pumped" ALUs (2 Operationen pro Taktzyklus)

- Simple Anweisungen (hinzufügen)

Sehr große "Pipeline"

- 20+Zyklus "Branch Penalty"
- ermöglicht sehr hohe Taktfrequenzen
- Langsamer als Pentium III mit gleicher Taktfrequenz

Prozessor Zusammenfassung

Designtechnik

- Benutzung eines einheitlichen "Frameworks" für alle Anweisungen
  - ✦ Ermöglicht den Anweisungen, die Hardwareressourcen untereinander zu teilen
- Kontrolllogik steuert Hardwareressourcen

Prozessor Zusammenfassung

Designtechnik

- Benutzung eines einheitlichen "Frameworks" für alle Anweisungen
  - ✦ Ermöglicht den Anweisungen, die Hardwareressourcen untereinander zu teilen
- Kontrolllogik steuert Hardwareressourcen

Operation

- Zustand wird in Speichern und Registern gehalten
- Berechnung wird von kombinatorischer Logik durchgeführt
- Taktung der Register/Speicher kontrolliert das allgemeine Verhalten

Prozessor Zusammenfassung

Designtechnik

- Benutzung eines einheitlichen "Frameworks" für alle Anweisungen
  - ✦ Ermöglicht den Anweisungen, die Hardwareressourcen untereinander zu teilen
- Kontrolllogik steuert Hardwareressourcen

Operation

- Zustand wird in Speichern und Registern gehalten
- Berechnung wird von kombinatorischer Logik durchgeführt
- Taktung der Register/Speicher kontrolliert das allgemeine Verhalten

Leistungssteigerung

- "Pipelining" erhöht den Datendurchsatz und verbessert die Ressourcennutzung
- Muss sichergehen, dass ISA Verhalten beibehalten wird

Die Speicherhierarchie

Themen:

- Speichertechnologien und Trends
- Lokalität der Referenzen
- "Caching" in der Speicherhierarchie

Random-Access Memory (RAM)

Hauptmerkmale

- RAM ist als Chip gepackt
- Grundspeichereinheit ist eine Zelle (ein Bit pro Zelle)
- Viele RAM Chips bilden einen Speicher

Random-Access Memory (RAM) II

Statischer RAM (SRAM)

- Jede Zelle speichert Bits mit einer 6-Transistor Schaltung
- Speichert Wert solange er mit Energie versorgt wird
- Relativ unanfällig für Störungen wie elektrische Brummspannungen
- Schneller und teurer als DRAM

Random-Access Memory (RAM) II

Statischer RAM (SRAM)

- Jede Zelle speichert Bits mit einer 6-Transistor Schaltung
- Speichert Wert solange er mit Energie versorgt wird
- Relativ unanfällig für Störungen wie elektrische Brummspannungen
- Schneller und teurer als DRAM

Dynamischer RAM (DRAM)

- Jede Zelle speichert Bits mit einem Kondensator und einem Transistor
- Der Wert muss alle 10-100 ms aufgefrischt werden
- Anfällig für Störungen
- Langsamer und billiger als SRAM

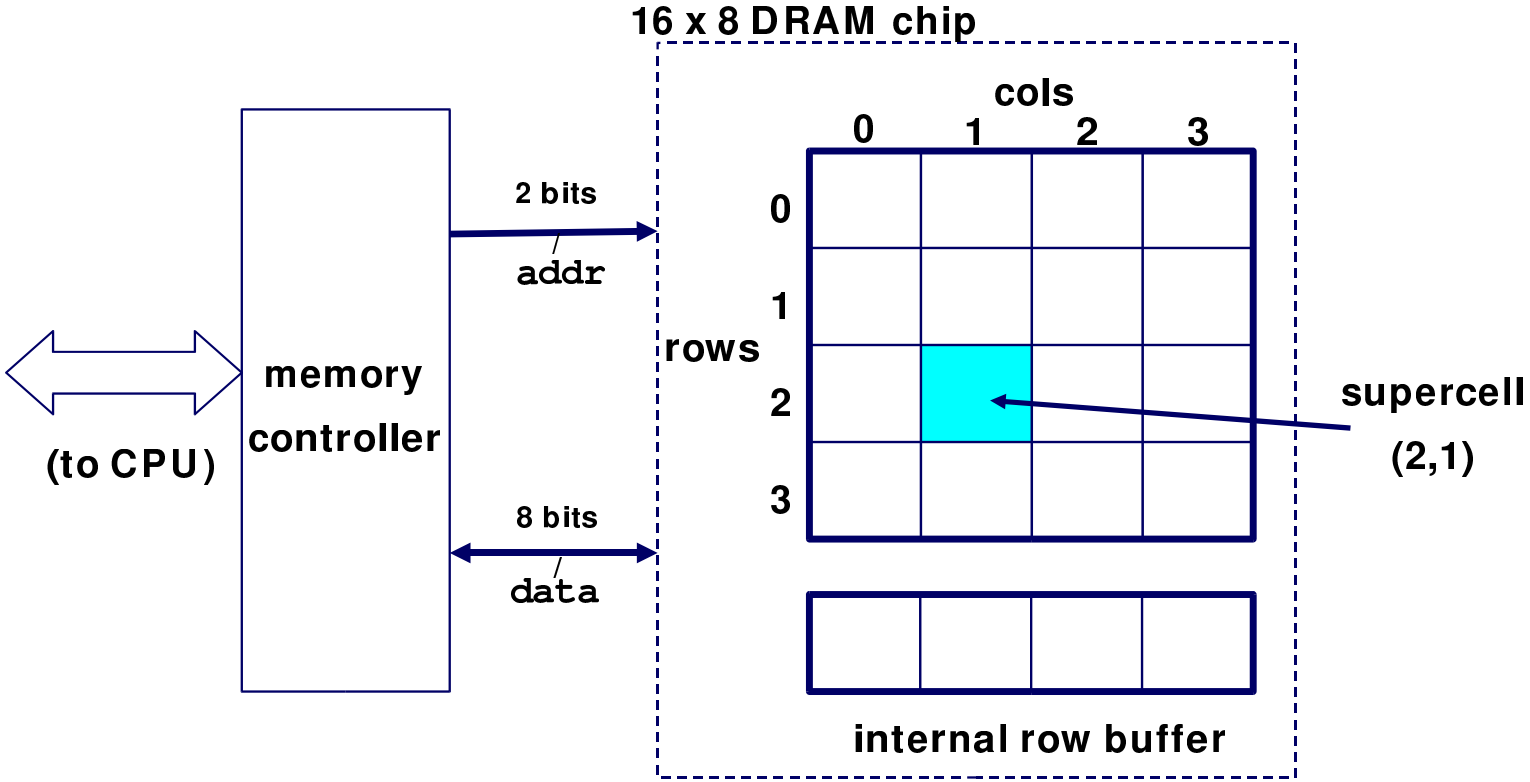
SRAM gegen DRAM Zusammenfassung

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers

Konventionelle DRAM Organisation

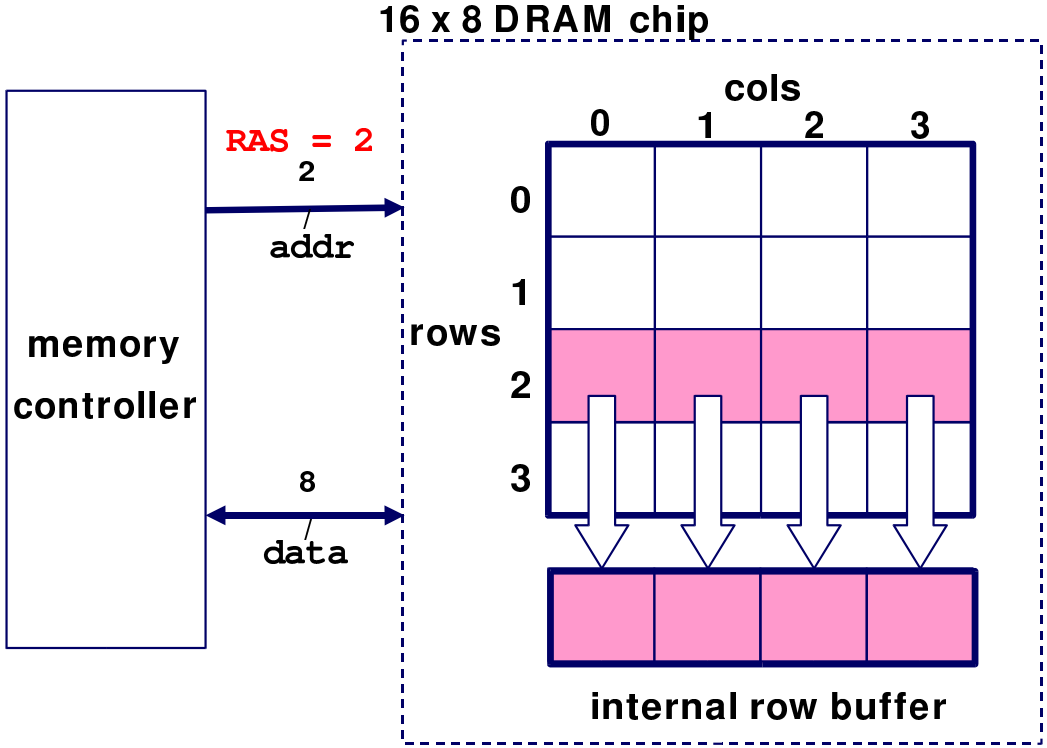
d x w DRAM:

- dw Summe aller Bits organisiert als d Superzellen mit w Bits



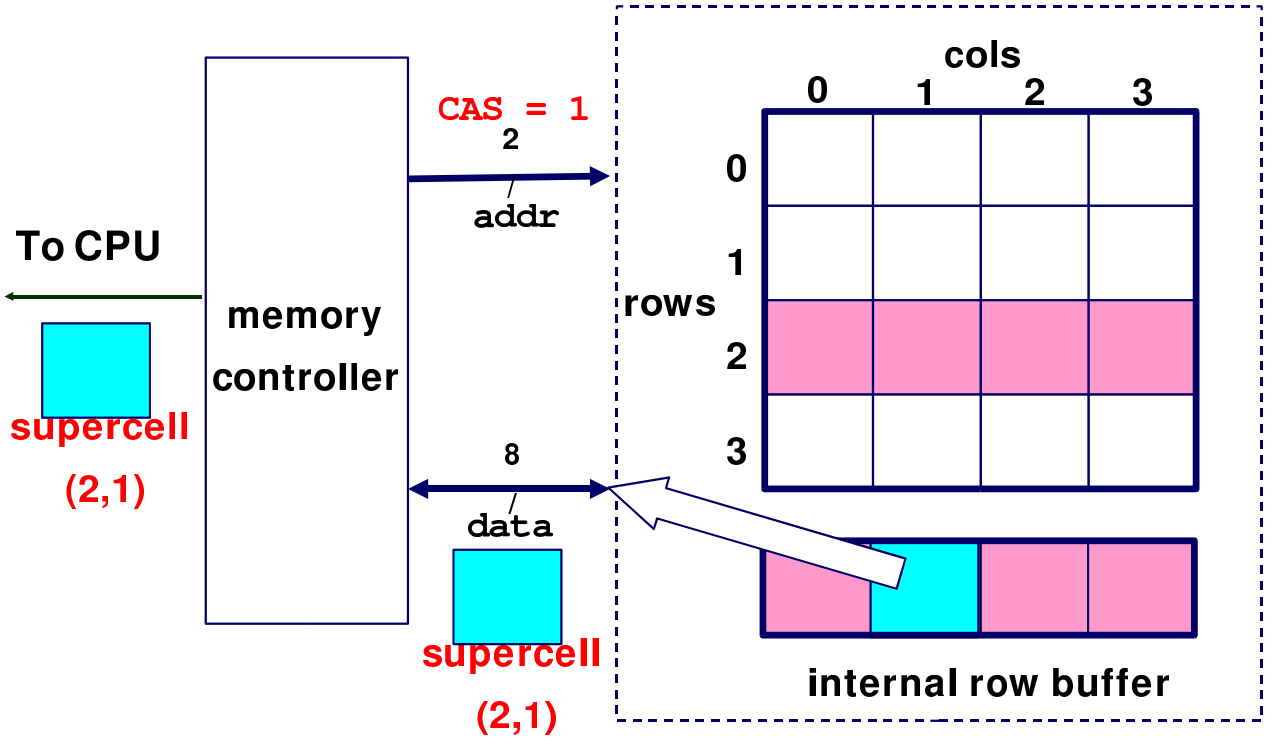
Lesen der DRAM Superzelle (2,1)

Schritt 1(a): "Row Access Strobe" (RAS) wählt Zeile 2.
Schritt 1(b): Zeile 2 wird aus dem DRAM Array in den Zeilenpuffer ("Row Buffer") kopiert.



Lesen der DRAM Superzelle (2,1)

Schritt 2(a): "Column Access Strobe" (CAS) wählt Spalte 1.
Schritt 2(b): Superzelle (2,1) wird aus dem Buffer ausgelesen und auf die Datenleitungen gelegt. Von dort gelangen die Daten zurück in die CPU



Erweiterte DRAMS

Alle erweiterten DRAMS sind um den konventionellen DRAM Kern herum aufgebaut

- "Fast Page Mode DRAM" (FPM DRAM)
  - Greift auf den Inhalt einer Speicherzeile zu mit [RAS, CAS, CAS, CAS, CAS]
    anstatt mit [(RAS, CAS), (RAS, CAS), (RAS, CAS), (RAS, CAS)].
- "Extended Data Out DRAM" (EDO DRAM)
  - Erweiterter FPM DRAM mit schneller aufeinanderfolgenden CAS Signalen
- Synchroner (taktgesteuerter) DRAM (SDRAM)
  - Betrieben mit steigender Taktflanke statt mit asynchronen Kontrollsignalen

Erweiterte DRAMS

Weitere erweiterten DRAMS

- "Double Data-Rate Synchronous DRAM" (DDR SDRAM)
  - SDRAM-Erweiterung die beide Taktflanken als Kontrollsignal benutzt
- Video RAM (VRAM)
  - Wie FPM DRAM, aber Output wird durch Verschieben des "Row Buffer" hergestellt
  - Zwei Ports (erlauben gleichzeitiges Lesen und Schreiben)

Nichtflüchtige Speicher

DRAM und SRAM sind flüchtige Speicher.

- Informationen gehen beim Abschalten verloren

Nichtflüchtige Speicher

DRAM und SRAM sind flüchtige Speicher.

- Informationen gehen beim Abschalten verloren

Nichtflüchtige Speicher speichern Werte selbst wenn sie abgeschaltet sind

- Allgemeiner Name ist “Read-Only-Memory” (ROM)
- Irreführend, da einige ROMs gelesen und verändert werden können

Nichtflüchtige Speicher

DRAM und SRAM sind flüchtige Speicher.

- Informationen gehen beim Abschalten verloren

Nichtflüchtige Speicher speichern Werte selbst wenn sie abgeschaltet sind

- Allgemeiner Name ist "Read-Only-Memory" (ROM)
- Irreführend, da einige ROMs gelesen und verändert werden können

Arten von ROMs

- Programmmierbarer ROM (PROM)
- "Erasable Programmable ROM" (EPROM) (Löschbarer, programmierbarer ROM)
- "Electrically Eraseable PROM" (EEPROM) (Elektrisch löschbarer PROM)
- Flash Speicher

Anwendungsbeispiel nichtflüchtige Speicher

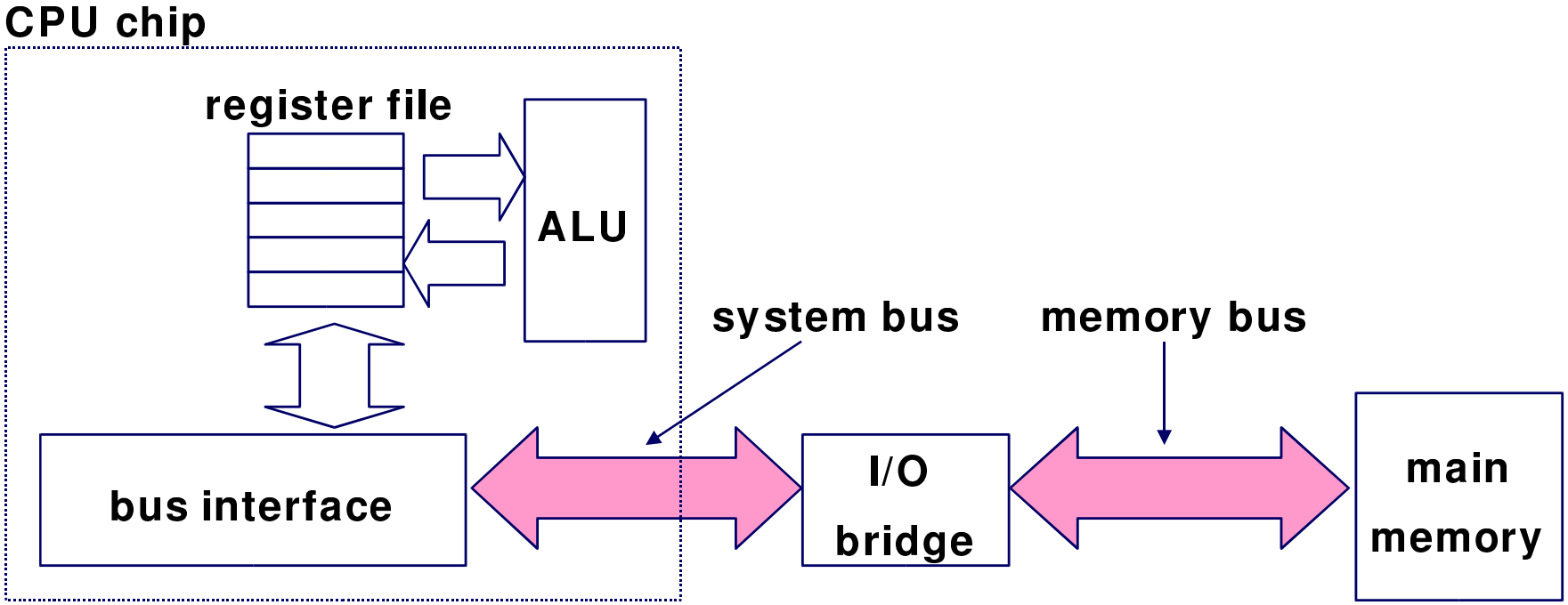
Firmware

- Programm wird in einem ROM gespeichert
  - Boot Zeitcode, BIOS (basic input/output system)
  - Grafikkarten, Festplattencontroller

Typische Busstrukturen zur Verbindung von CPU und Speicher

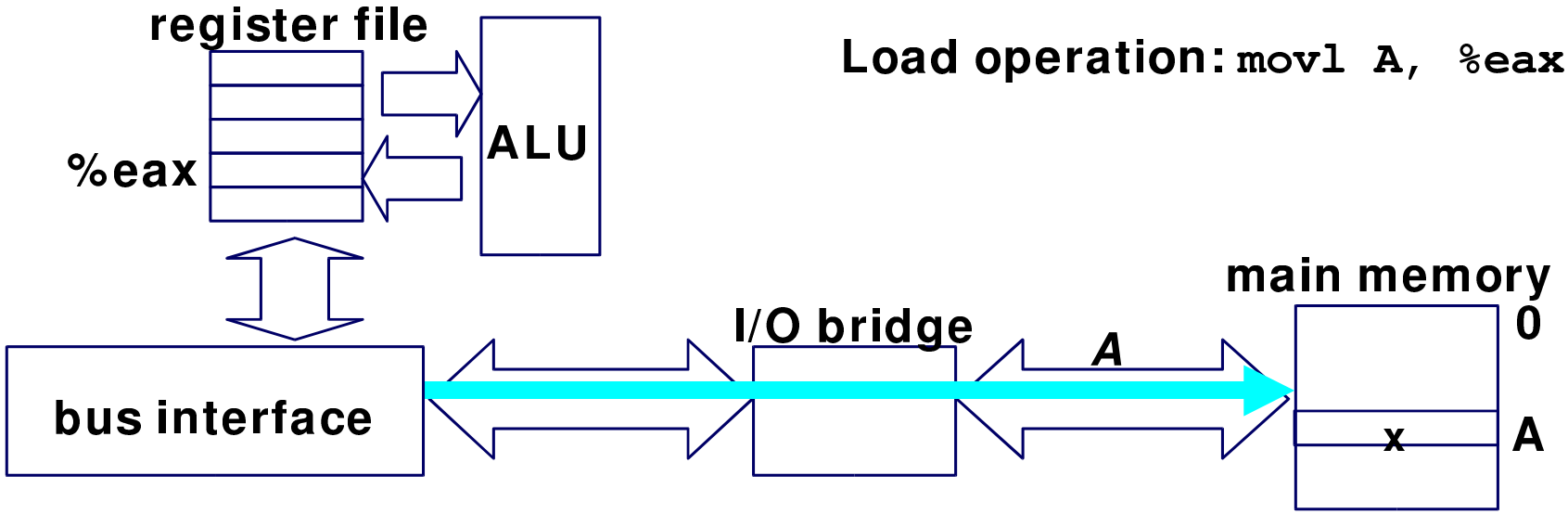
Ein Bus ist eine Sammlung paralleler Drähte die Adressen, Daten und Kontrollsignale übertragen.

Busse werden üblicherweise von mehreren Geräten genutzt.



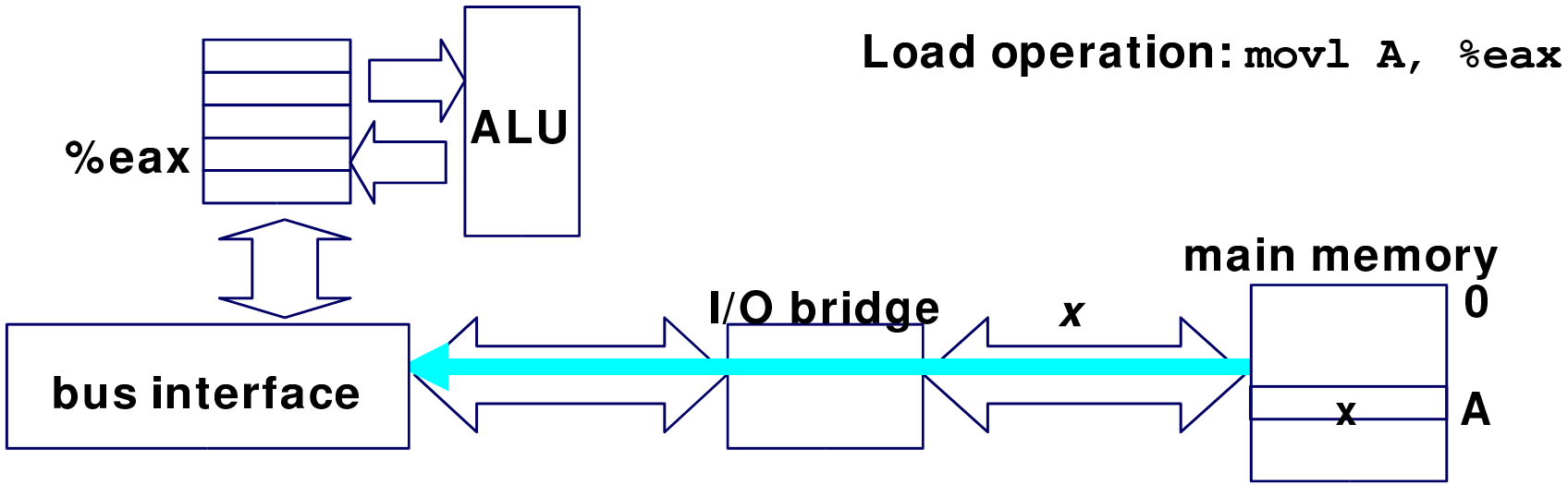
Speicherlesevorgang (1)

CPU legt Adresse A auf den Speicherbus.



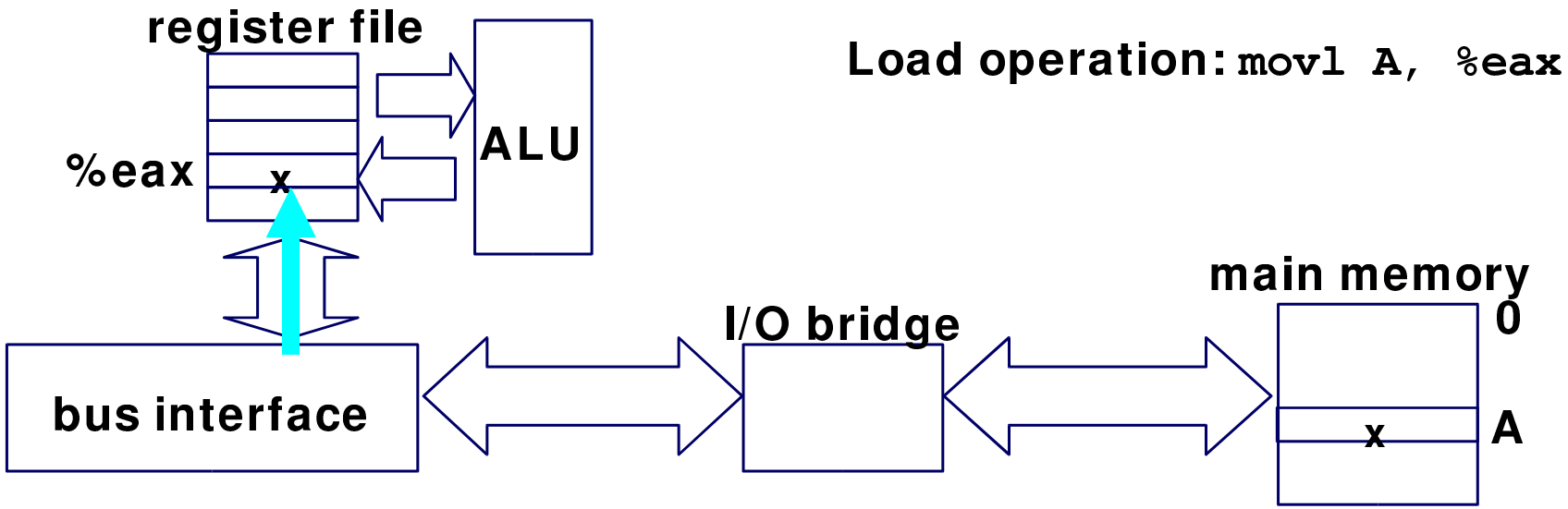
Speicherlesevorgang (2)

Hauptspeicher liest Adresse A vom Speicherbus, ruft das Wort x ab und legt es auf den Bus.



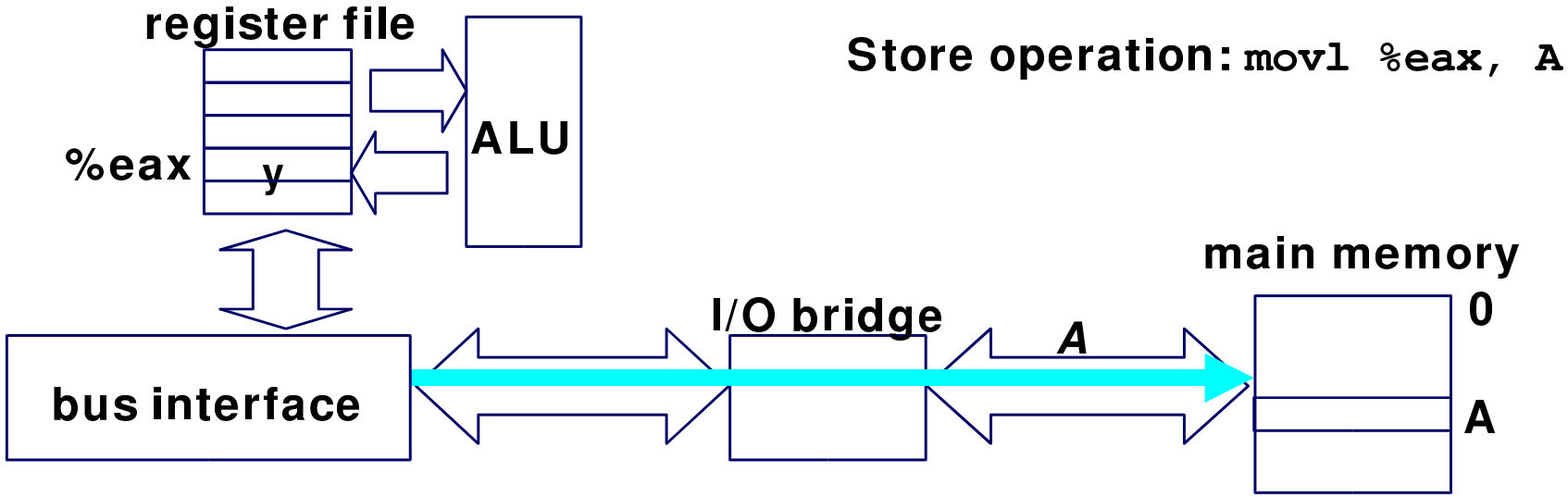
Speicherlesevorgang (3)

CPU liest Wort x vom Bus und kopiert es ins Register %eax.



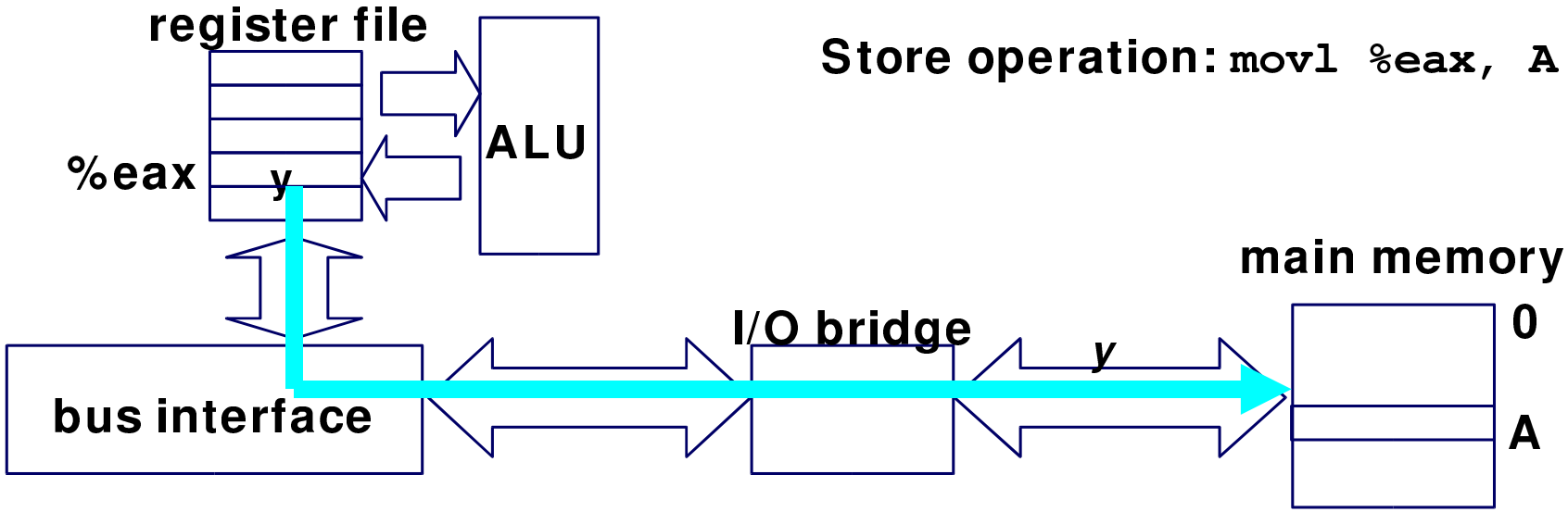
Speicherschreibvorgang (1)

CPU legt die Adresse A auf den Bus. Der Hauptspeicher liest sie und wartet auf die Ankunft des entsprechenden Datenworts.



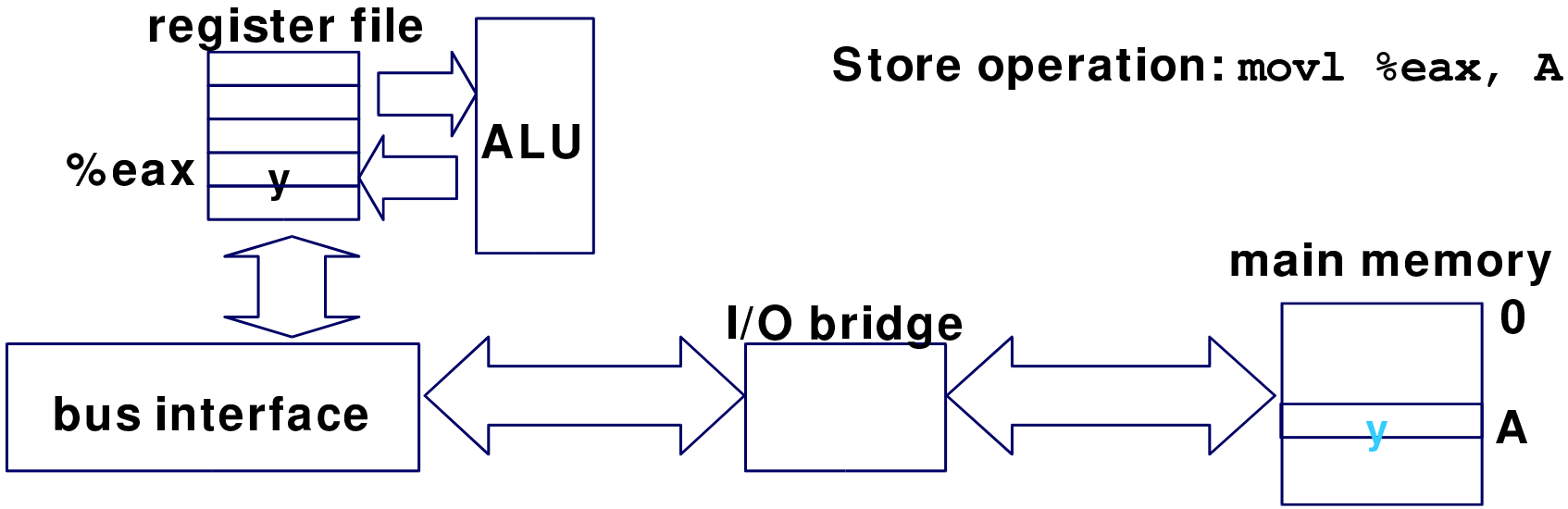
Speicherschreibvorgang (2)

CPU legt Datenwort y auf den Bus.



Speicherschreibvorgang (3)

Hauptspeicher liest Datenwort y vom Bus und speichert es unter Adresse A ab.



Ergänzende Literatur

Zur Rechnerarchitektur (Teil 2, 6. Termin):

Literatur

[1] Randal E. Bryant and David O'Hallaron. Computer systems. pages 317–360, 455–466. Pearson Education, Inc., New Jersey, 2003.