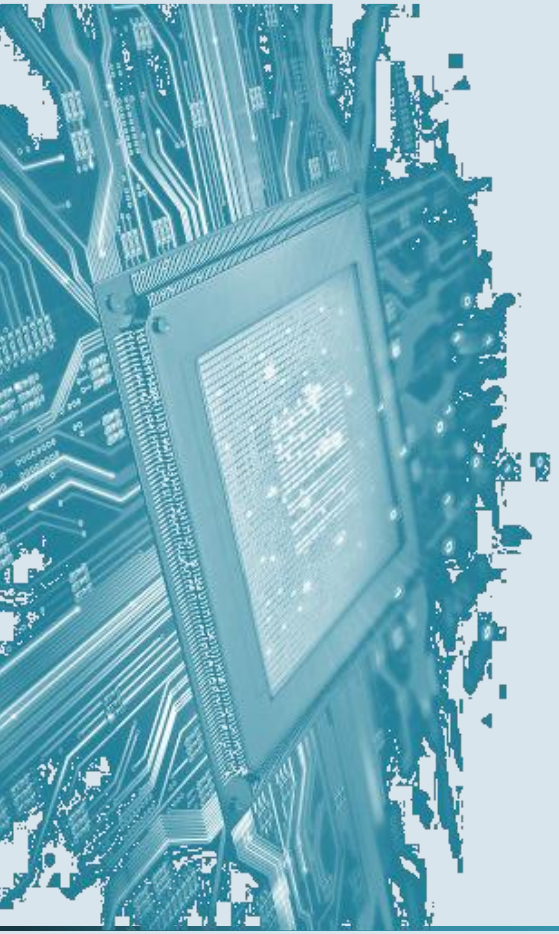


Rechnerarchitektur (AIN 2)

SoSe 2021

Kapitel 4

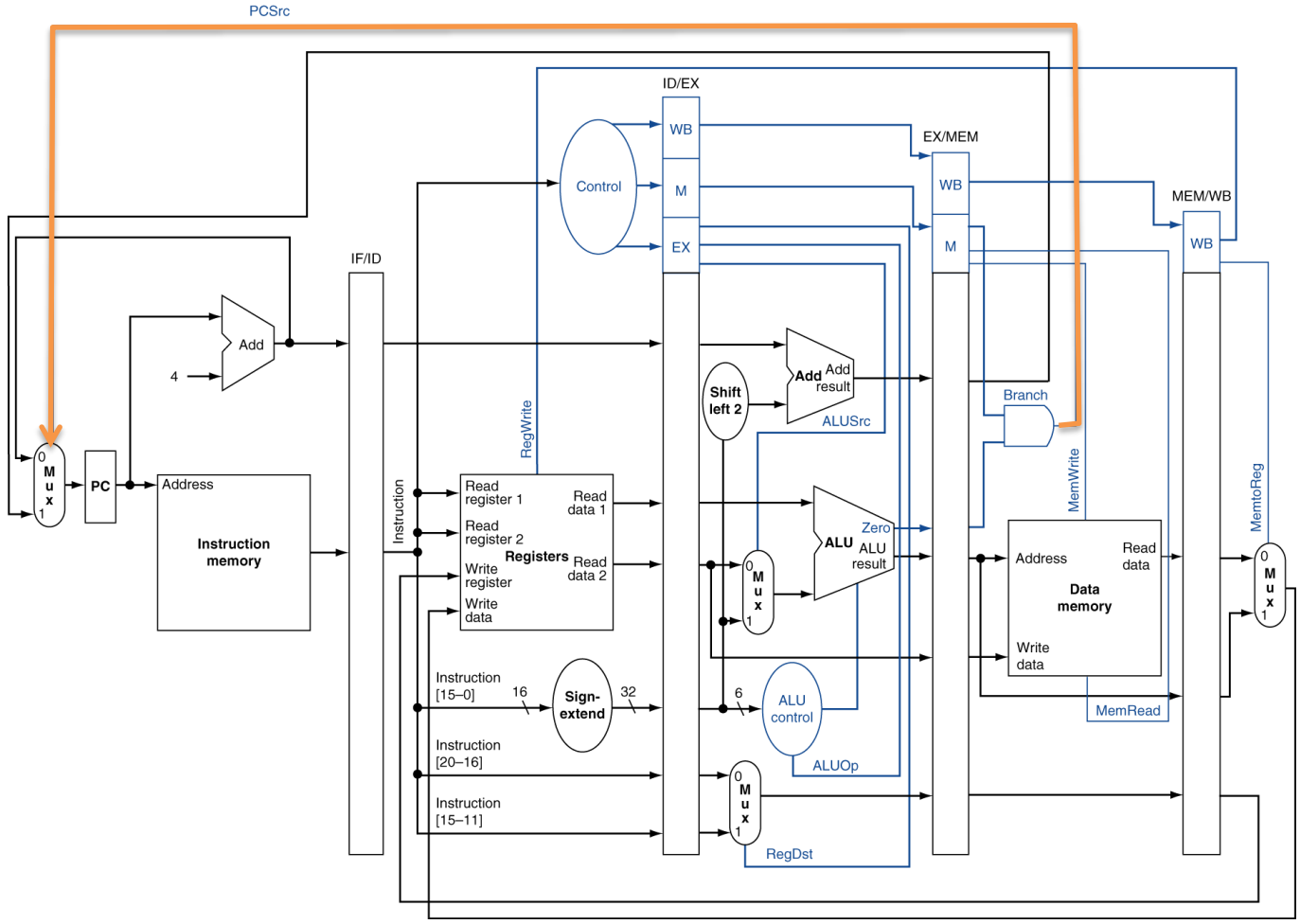
Multi-Cycle CPU – Pipeline-Architekturen

Prof. Dr.-Ing. Michael Blaich
mblaich@htwg-konstanz.de

Kapitel 4: Multi-Cycle CPU – Pipeline-Architekturen

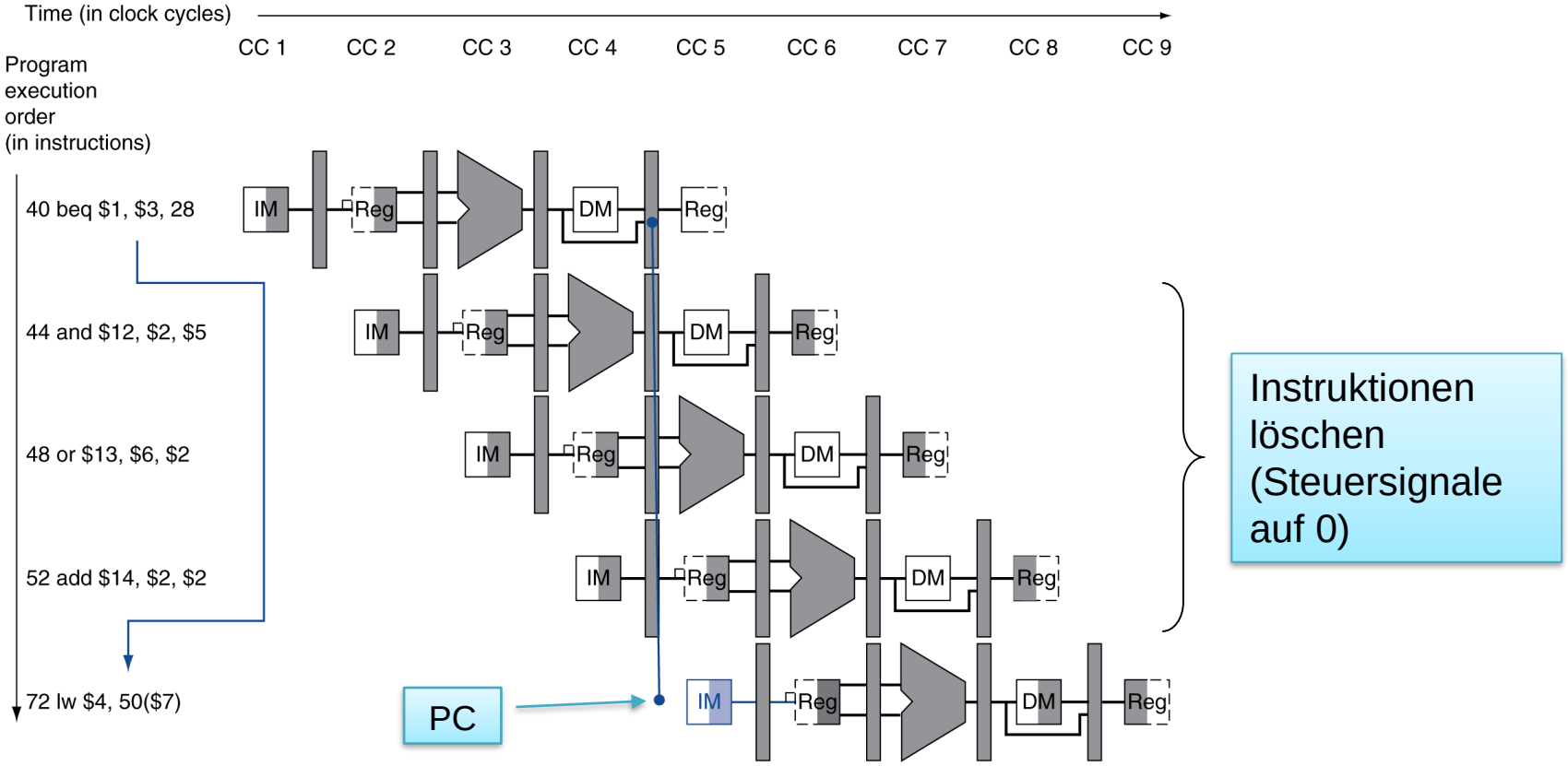
Branches in der MEM-Stage

In der bisherigen Pipeline wird im Falle eines Sprunges der PC in der MEM-Stage geschrieben.



Branches in der MEM-Stage

- Instruktionen nach einer Branch-Instruktion werden ausgeführt
- Im Falle eines Sprunges
  - wird der PC in der MEM-Stage auf die Sprungadresse gesetzt
  - IF/ID, ID/EX und EX/MEM werden auf NOP gesetzt
- Sprung kostet 3 Takte ➔ Branches in der ID-Stage und Branch-Prediction



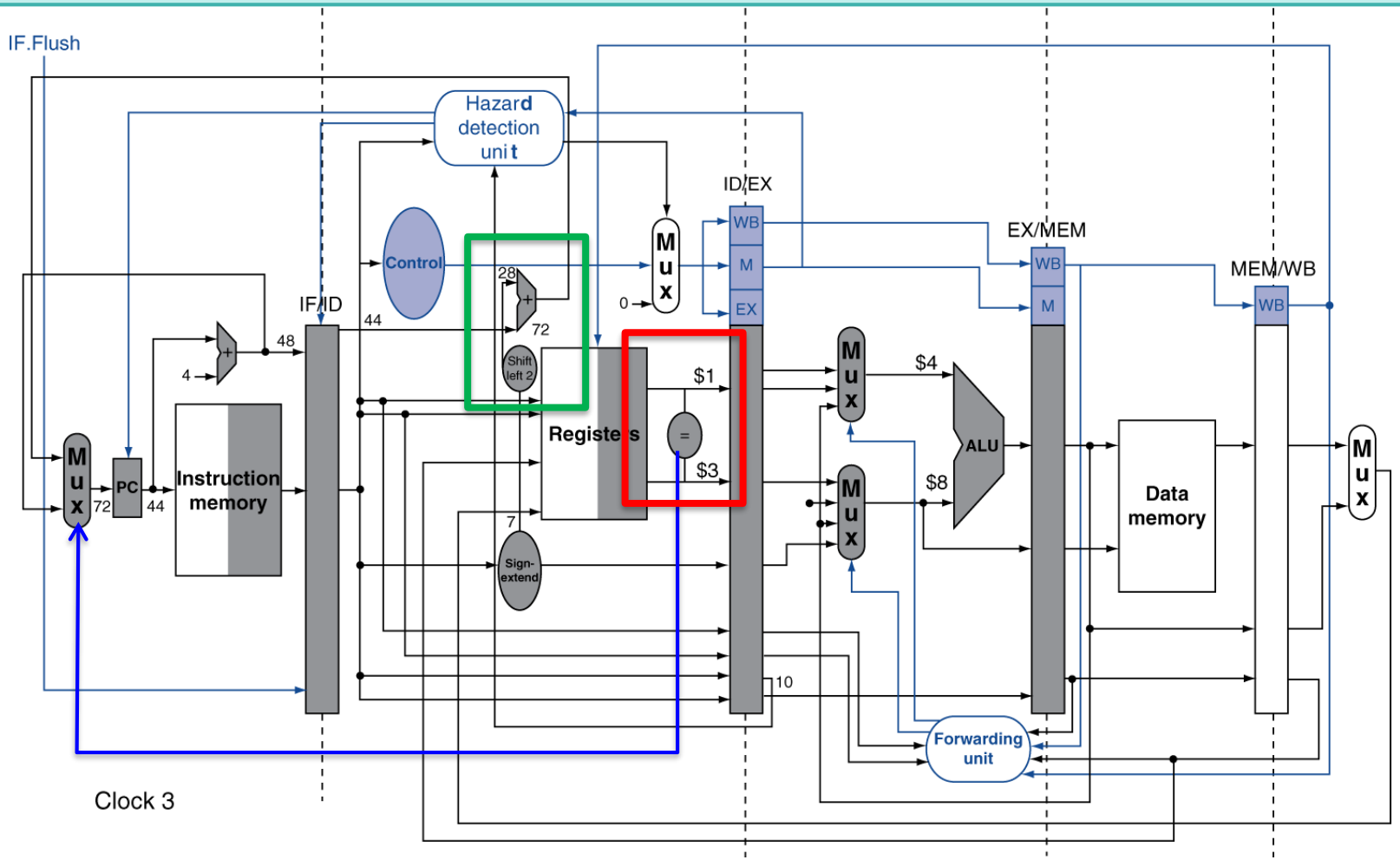
Früherkennung von Sprüngen

- Zusätzliche Hardware zur Bestimmung des Sprungziels in der ID Stage
  - Zusätzlicher Addierer für die Ziel-Adresse
  - Zusätzliche Hardware-Komponente zum Vergleichen zweier Register
- Beispiel für die nächsten Folien:

```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7 ➔ Spungziel: 44 + 4 · 7 = 72
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

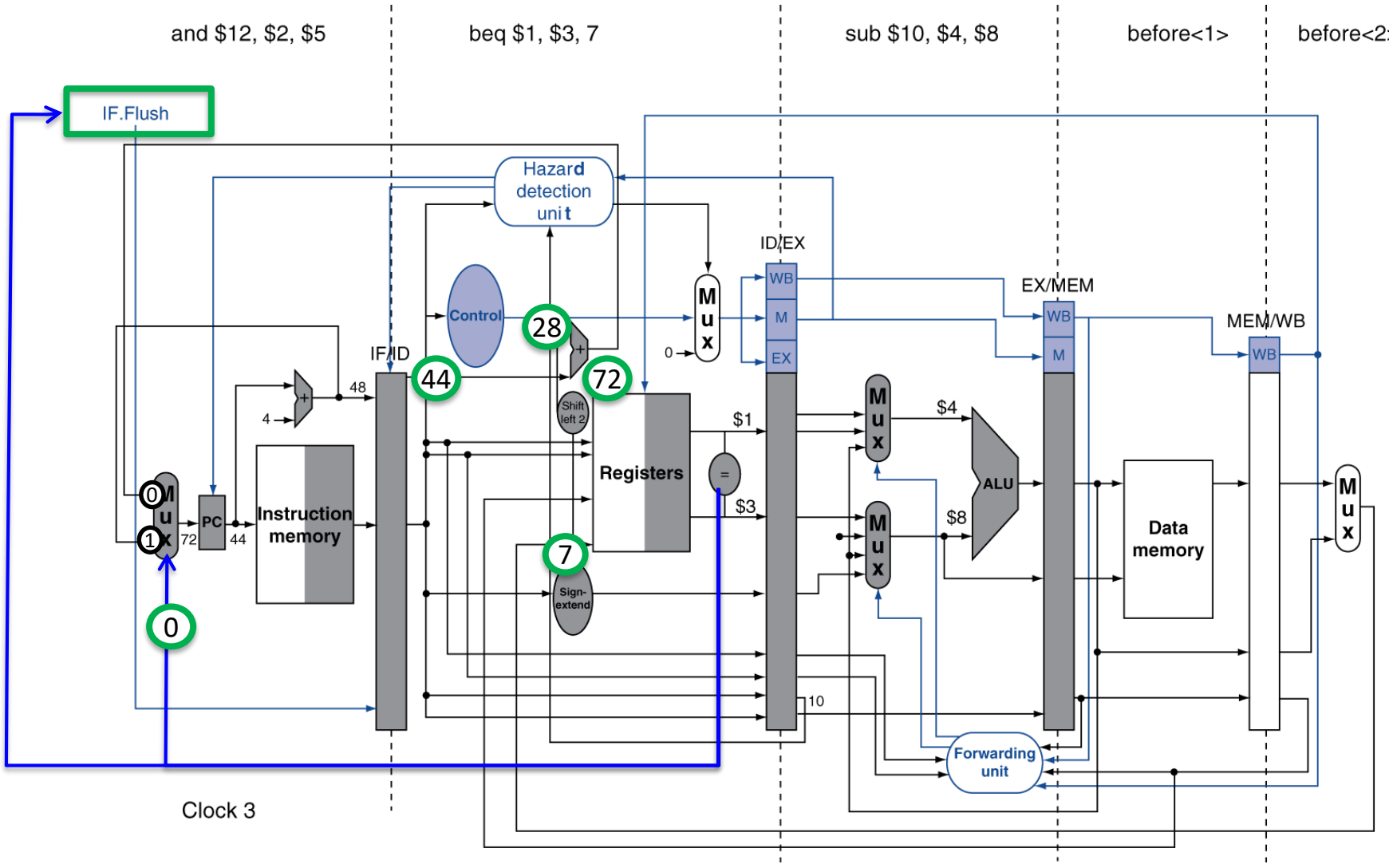
Zusätzliche Hardware in ID-Stage

- Sprungadresse: Shift Left 2 und Addition zu PC+4
  - Eingang in MUX für nächsten PC
- Sprungbedingung: Vergleich der Registeroperanden
  - wird mit PCSrc kombiniert und dient als Steuersignal für PC-MUX



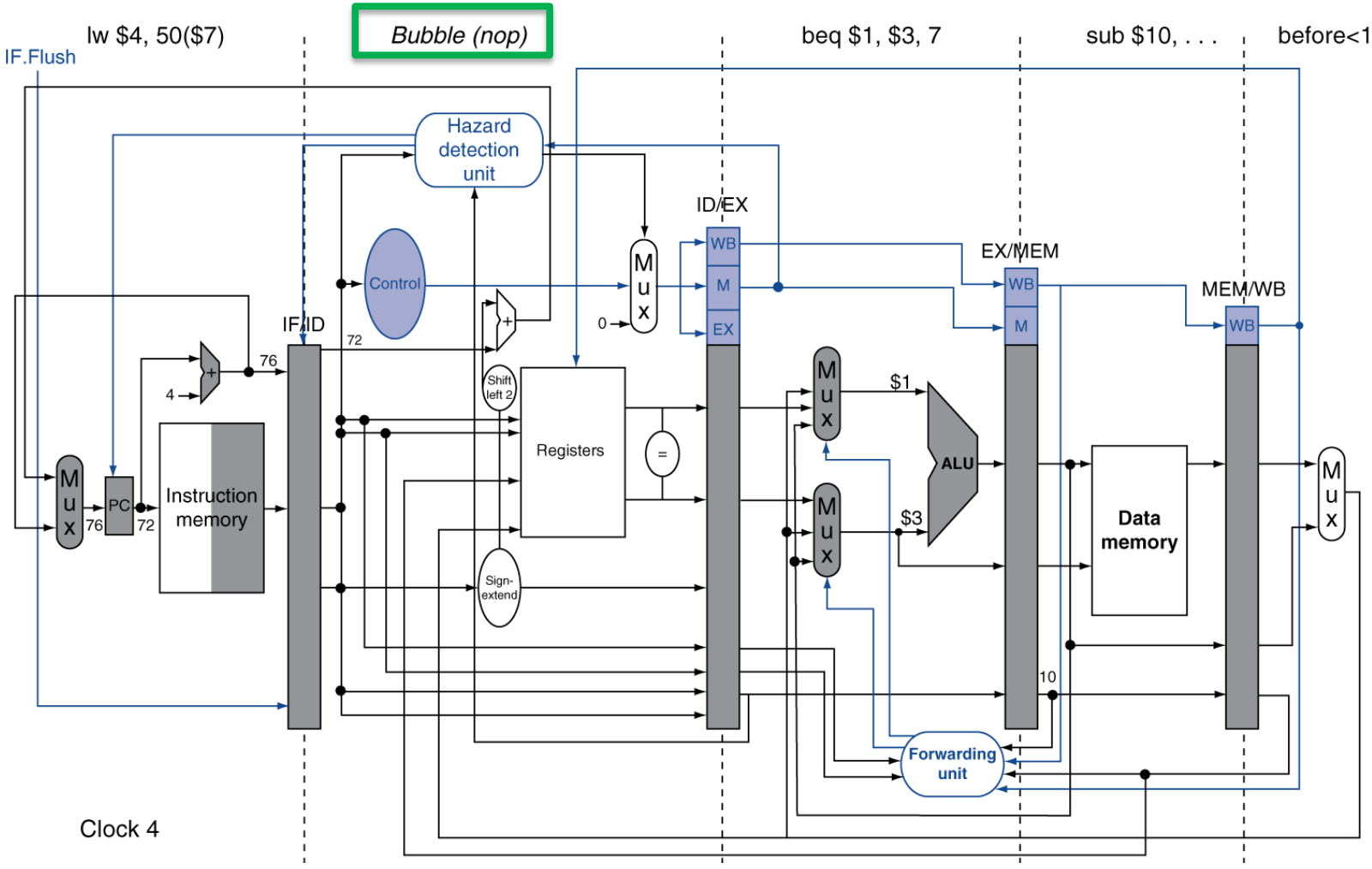
Beispiel: Sprung wird ausgeführt

- Nächster PC=Sprungadresse: 44+4*7=44+28=72
- IF.Flush=True: NOP in IF/ID-Register



Beispiel: Sprung wird ausgeführt (Takt 4)

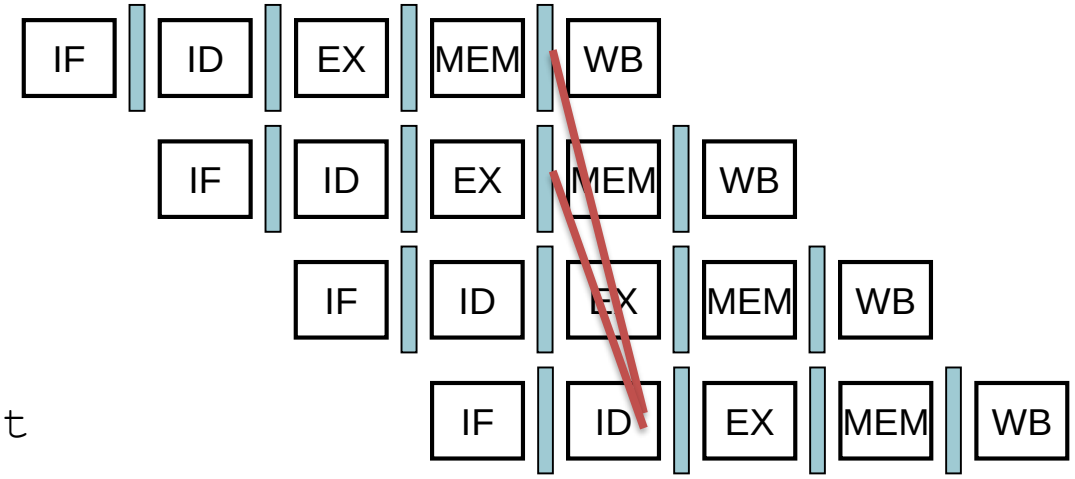
- Durch das „flushen“ des IF/ID-Registers entsteht in ID eine Bubble
- Instruktion an Adresse 72 wird geladen, PC wird auf 76 erhöht



Data Hazards für Branches

- Der Vergleich der Registeroperanden wird in der ID-Stage durchgeführt, folglich werden die Registerinhalte in der ID-Stage benötigt
  - bislang wurde mit Registeroperanden erst in der EX-Stage gerechnet
- Ergebnisse von Instruktionen, die zwei oder drei Schritte früher gestartet wurden, können benötigt werden

```
add $1, $2, $3
add $4, $5, $6
 …
beq $1, $4, target
```

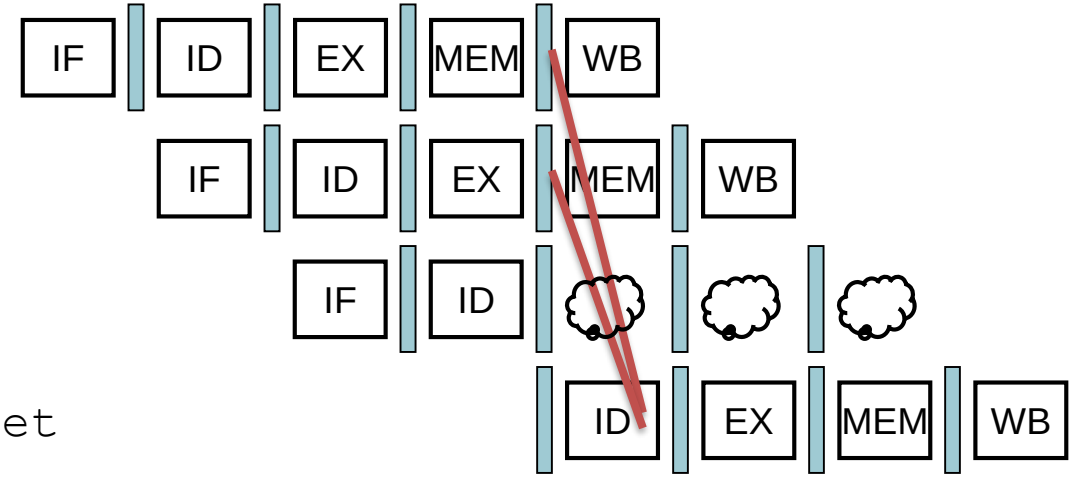


- Lösung: Forwarding zum Vergleichsoperator in der ID-Stage

Beispiel: Stalling bei Load-Branch-Hazards

- Betrachten die folgende Situation:
  - Instruktion n ist Load-Instruktion
    - lädt Speicherinhalt in Register $1
  - Instruktion n+2 ist eine Branch-Instruktion
    - führt Vergleich mit Register $1 durch
  - Branch muss einen Takt warten, bis Forwarding vom MEM/WB-Register in die ID-Stage möglich ist

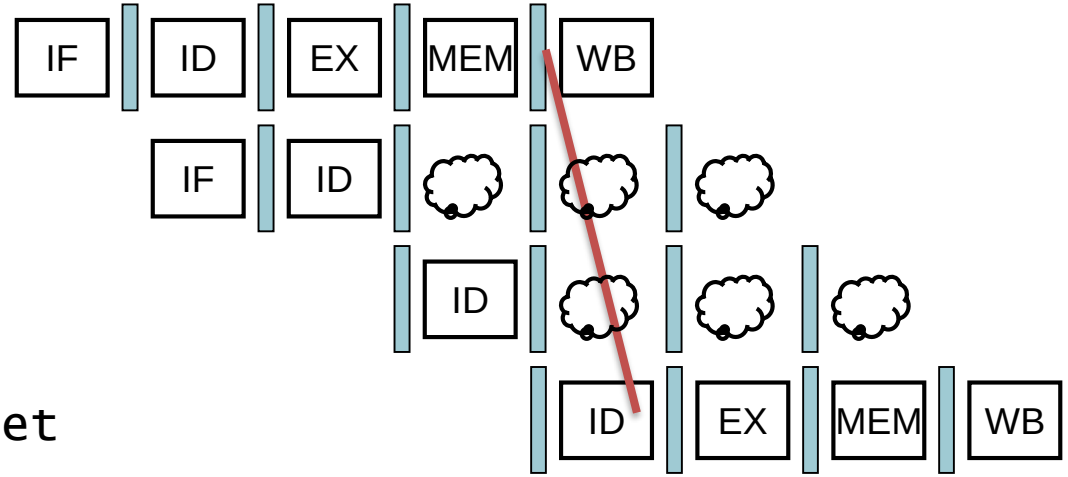
```
lw   $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



Stalling bei Load-Branch-Hazards

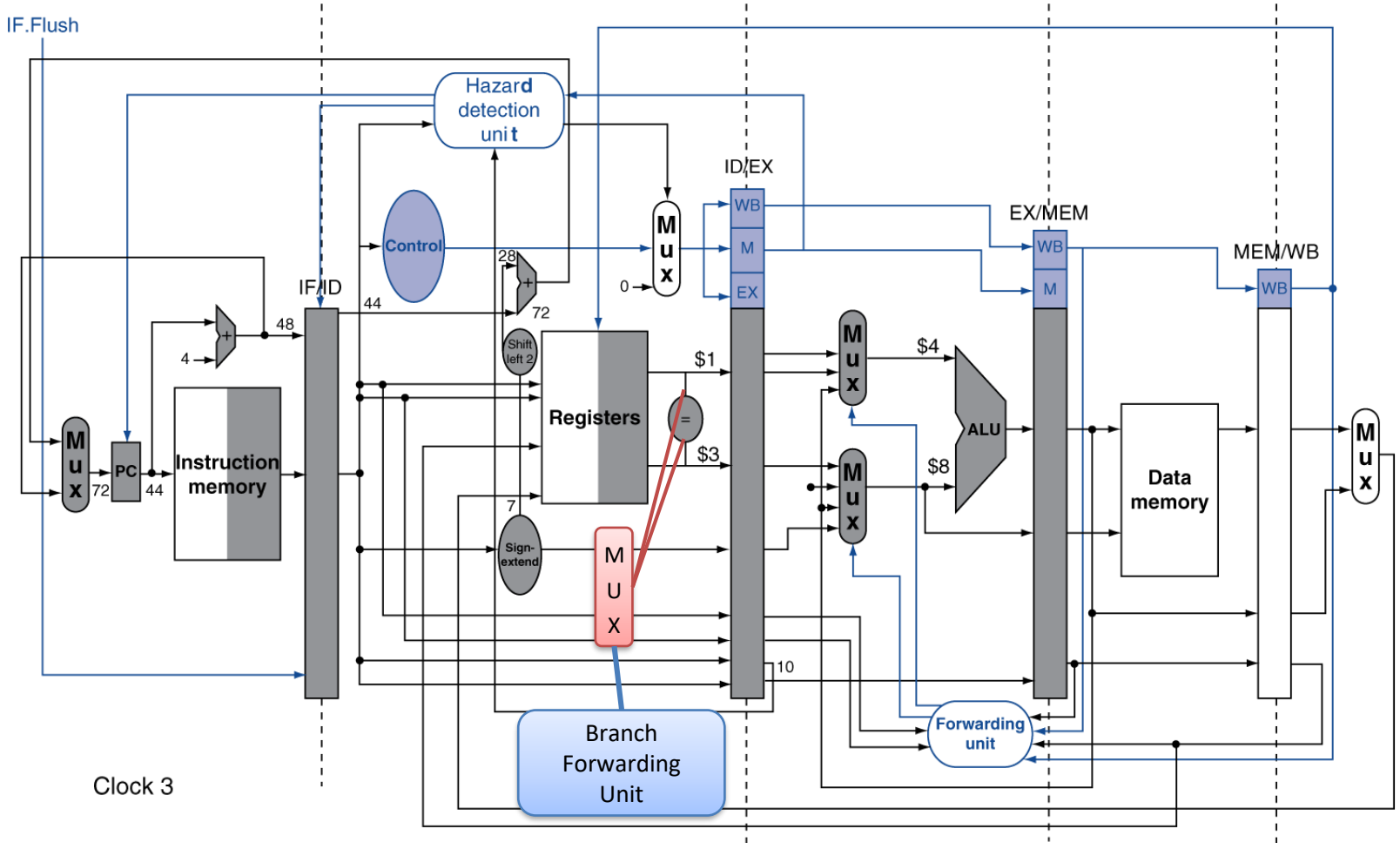
- Betrachten die folgende Situation:
  - Instruktion n ist Load-Instruktion
    - lädt Speicherinhalt in Register $1
  - Instruktion n+1 ist eine Branch-Instruktion
    - führt Vergleich mit Register $1 durch
  - Branch muss zwei Takte warten, bis Forwarding vom MEM/WB-Register in die ID-Stage möglich ist

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```



Branch Forwarding und Hazards?

Benötigen eine Branch Forwarding Unit und zwei Multiplexer für die beiden Vergleichsoperanden.



Kapitel 4: Multi-Cycle CPU – Pipeline-Architekturen

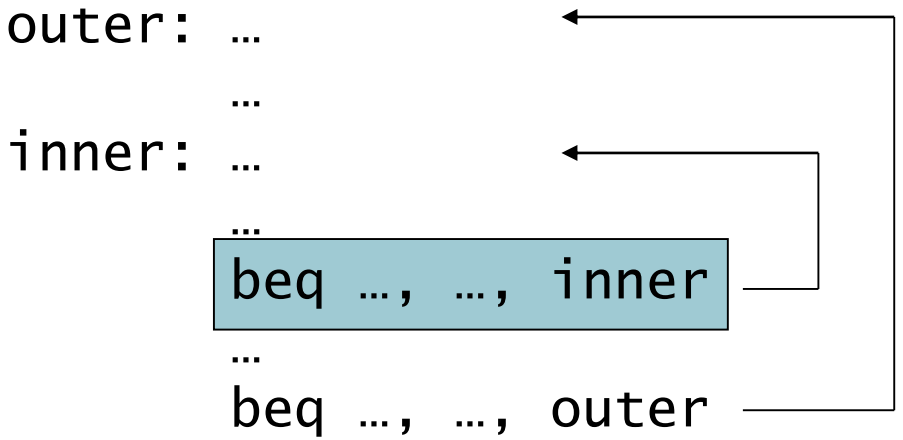
Dynamische Sprung-Prädiktion

- In Pipelines mit vielen Stages oder in superskalaren Architekturen (Kapitel 6) sind Performance-Einbrüche durch Branches signifikant
- Lösung: dynamische Sprung-Prädiktion
  - Branch prediction buffer (aka branch history table)
    - Sprung-Vorhersage-Speicher/Spungverlaufstabelle
    - wird für jede Sprunginstruktion erstellt
    - enthält Historie der letzten Sprungentscheidungen (genommen/nicht genommen, taken/not taken)
  - Einfache Realisierung: 1-Bit-Prediction-Buffer
    - Inhalt: ein Bit (taken/not taken)
      - letzter Sprung genommen oder nicht genommen
    - Ausführung eines Sprungs
      - Eintrag aus dem Branch-Prediction-Buffer holen
      - Instruktion entsprechend der Vorhersage laden
      - Im Fehlerfall, Instruktionen aus der Pipeline löschen und Bit im Branch-Prediction-Buffer kippen

Nachteile der 1-Bit-Prediction-Buffer

1-Bit-Prediction bedeutet, dass das Ergebnis der letzten Entscheidung einer Branch-Instruktion gespeichert und bei der nächsten Ausführung wieder „versucht“ wird. Diese Realisierung kann helfen, ist aber zu einfach.

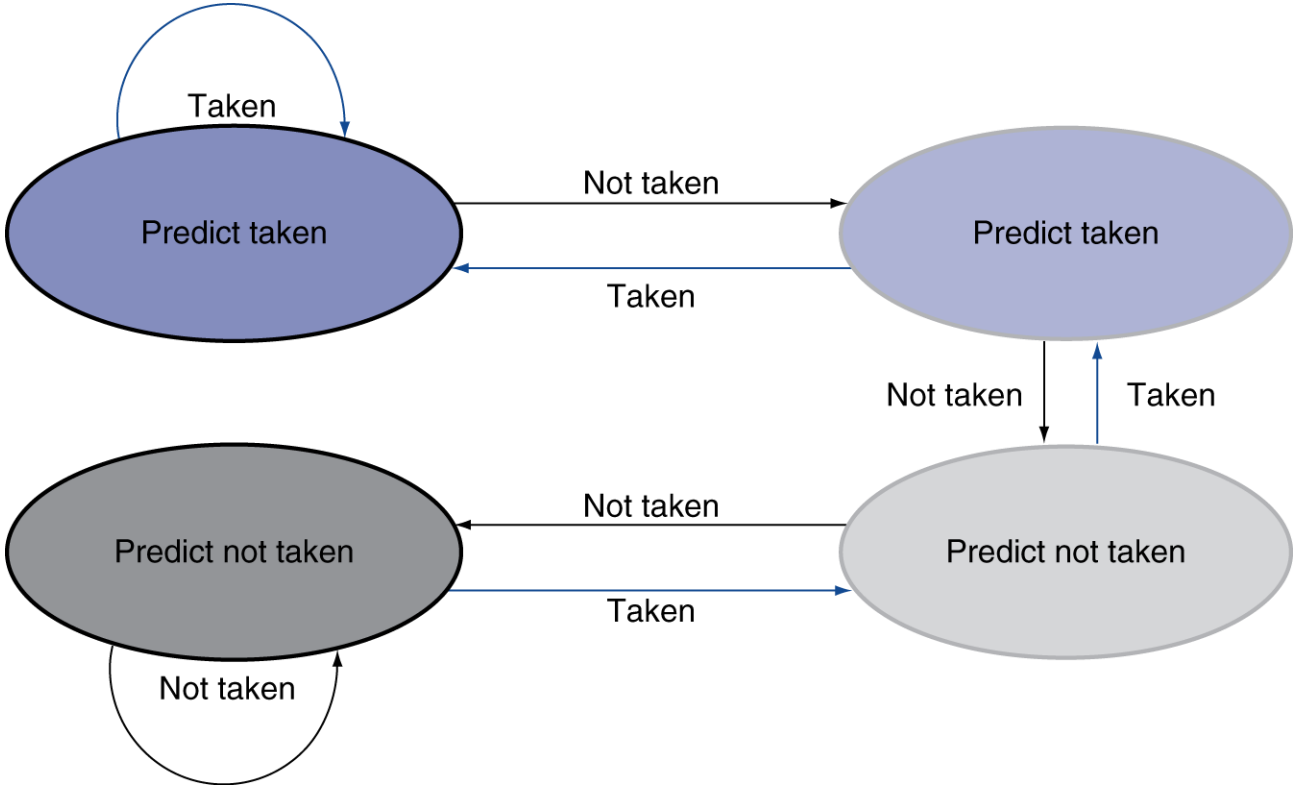
- Beispiel: Fehlentscheidungen bei doppelter Schleife mit Rückwärtssprüngen



- Zwei Fehlentscheidungen bei Sprung am Ende der inneren Schleife pro Durchlauf der äußeren Schleifen:
  - Fehlentscheidung bei letztem Durchlauf der inneren Schleife
  - Fehlentscheidung bei erstem Durchlauf der inneren Schleife bei wiederholtem Durchlauf der äußeren Schleifen

2-Bit Tabelle

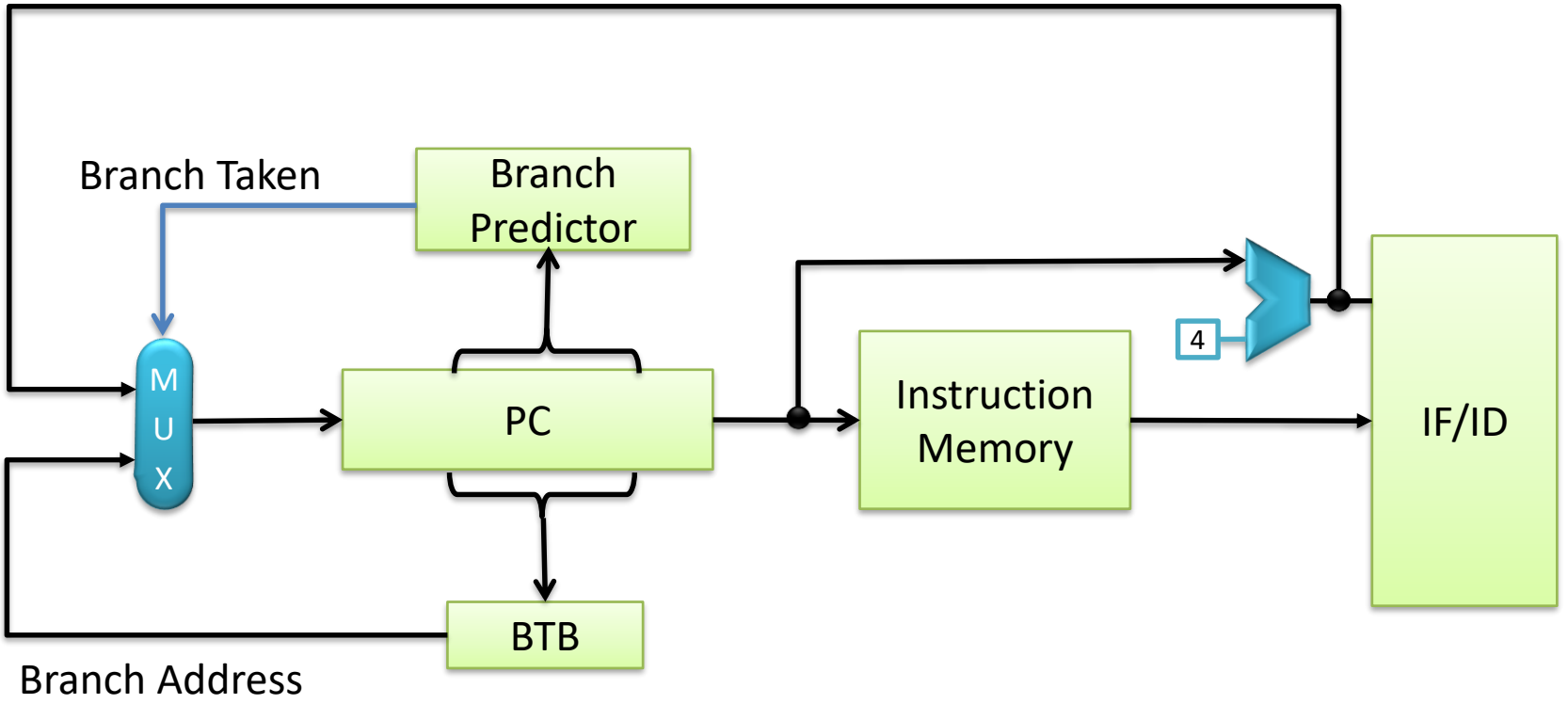
- Sprungtabelle hat 1-Bit-Gedächtnis
  - nach mindestens zwei gleichen Entscheidungen (z.B. taken/taken), ändert eine andere Entscheidung (not taken) die Vorhersage nicht
  - Änderung erst nach zwei falschen Vorhersagen oder bei Oszillation
- Zustandsübergangsdiagramm:



Branch Target Buffer

- Eine Vorhersage der Sprungadresse bringt nur einen Vorteil, wenn gleichzeitig auch die entsprechende Sprungadresse bekannt ist
  - in der MIPS-Pipeline werden Sprungadresse und Sprungbedingung in der ID Stage bestimmt
  - auch wenn die Sprungvorhersage in der IF-Stage durchgeführt wird, kann der PC erst in der ID-Stage gesetzt werden, da dort die Sprungadresse bestimmt wird
- Branch Target Buffer (BTB)
  - Speichern von Sprungadressen pro Program-Code-Adresse
  - Lesen der Sprungadresse aus BTB während IF
  - Indizierung durch Low-Level-Bits des PC
- Branch Prediction Buffer + Branch Target Buffer:
  - für jede Branch-Instruktion werden die Sprunghistorie sowie die Sprungadresse gespeichert

IF Stage mit Branch Prediction



- Aliasing
  - Branch Predictor und Branch Address können nicht für jede Programmadresse gespeichert werden
  - stattdessen werden z.B. nur 1024 Spungziele und Sprungvorhersagen gespeichert, die Bits 2-11 des Programm Counters adressiert werden
  - dadurch teilen sich mehrere Programmzeilen einen BTB und Branch-Predictor Eintrag
  - wurde durch die Spectre-Attacke ausgenutzt

Branch Vorhersage – Aktuelle Technik

- Fehlvorhersagen sind teuer
  - e.g. Intel Haswell: 15-20 Takte
- Aufwändige Branch-Vorhersage
  - Vorhersage abhängig von Branch-Historie
  - Beispiel:
    - Muster: 00110010… 1:Taken, 0: Not Taken
    - 1-Bit-Zähler pro Möglichkeit der letzten beiden Sprungentscheidungen

Letzte Sprünge	Zähler Stand (Taken/Not Taken)						
00	x	x	1	1	1	1	1+
01	x	x	x	1	1	1	1
10	x	x	x	x	x	0	0
11	x	x	x	x	0	0	0
Branch	0	0	1	1	0	0	1

  - Zusätzlich: Speichern der Loop-Länge
    - 0000001: 6 Schleifendurchläufe mit „Not Taken“, dann „Taken“

Branch Vorhersage – Beispiel

- Muster: 00110010… 1:Taken, 0: Not Taken
- 1-Bit-Zähler pro Möglichkeit der letzten beiden Sprungentscheidungen

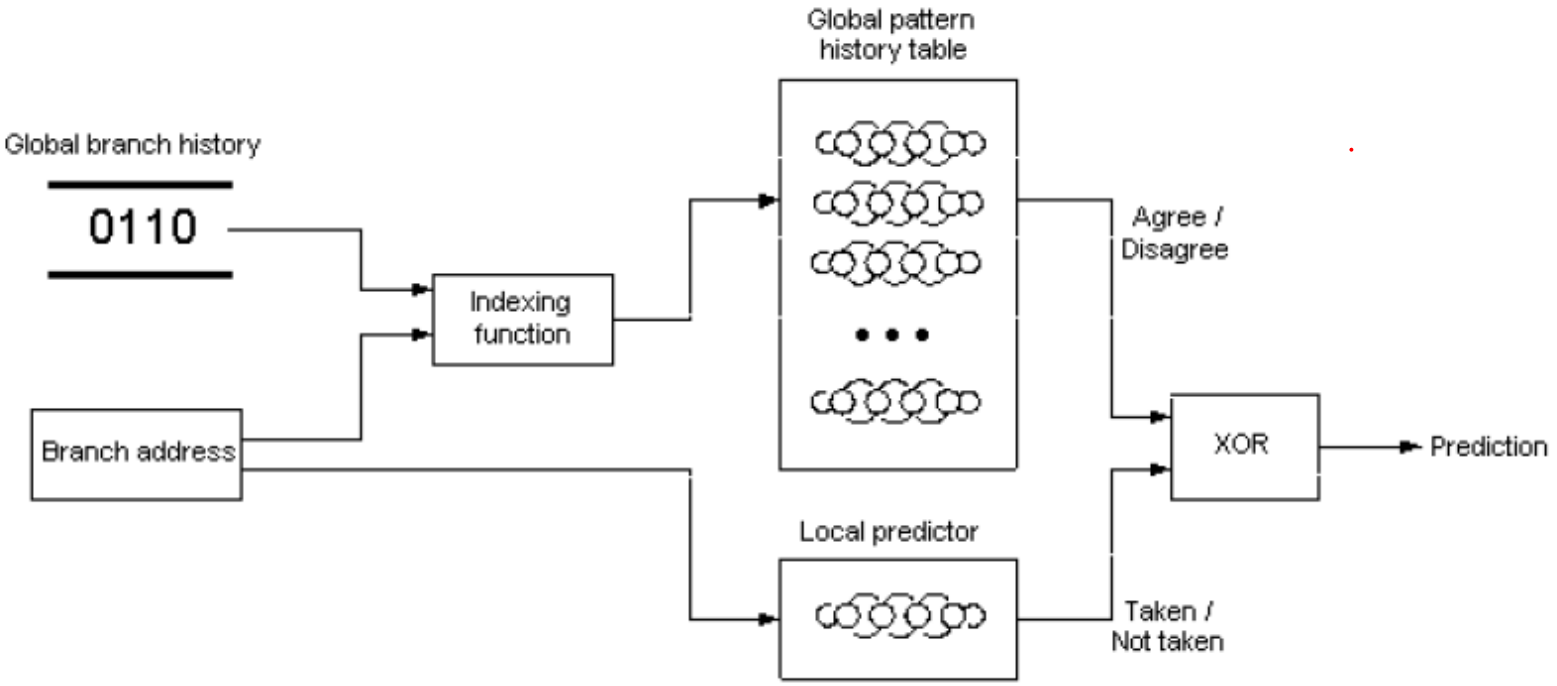
Vorhersage „Sprung“, falls die letzten beiden Entscheidungen „Kein Sprung“ waren.

Vorhersage „Kein Sprung“, falls die letzten beiden Entscheidungen „Sprung“ und „Kein Sprung“ waren.

Letzte Sprünge	Zähler Stand (Taken/Not Taken)						
00	x	x	1	1	1	1	1+
01	x	x	x	1	1	1	1
10	x	x	x	x	x	0	0
11	x	x	x	x	0	0	0
Branch	0	0	1	1	0	0	1

Branch Vorhersage – Aktuelle Technik

- Speichern der letzten N Branches
  - lokaler Buffer: letzte N Entscheidungen der Branch-Instruktion, kleines N
  - globaler Buffer: letzte N Branch-Entscheidungen im Programm, großes N
- 1- oder 2-Bit-Zähler für alle möglichen N-Branch Muster
  - globales Muster akzeptiert oder invertiert lokale Entscheidung



Kapitel 4: Multi-Cycle CPU – Pipeline-Architekturen

Exceptions und Interrupts

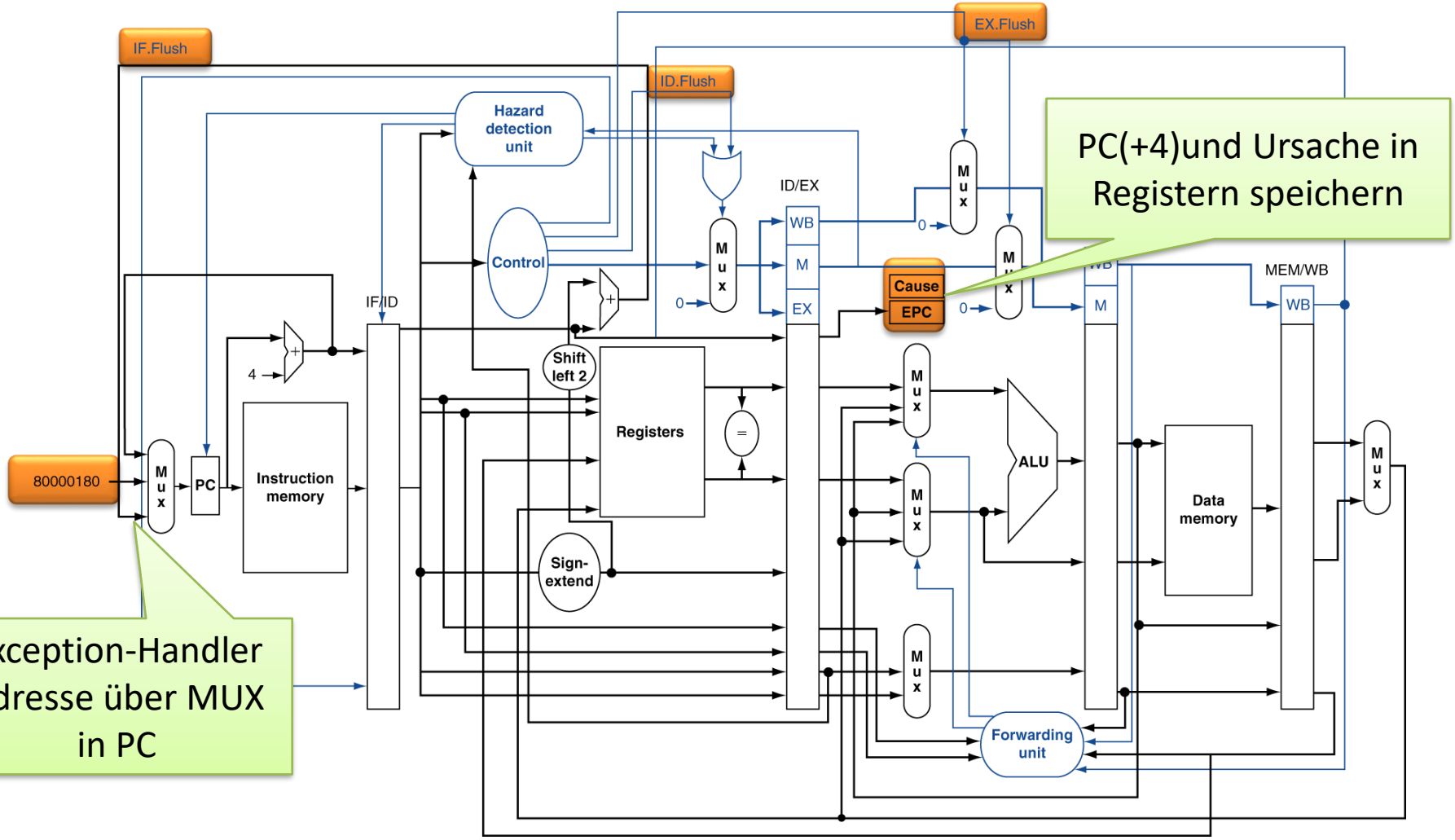
- Bisher: Ablauf der Pipeline planbar
- Exceptions und Interrupts (Unterbrechungen) sind “unplanmäßige” Ereignisse, die eine Anpassung des Kontrollflusses erforderlich machen
  - Terminologie je nach ISA unterschiedlich
- MIPS Terminologie:
  - Exception: tritt innerhalb der CPU auf
    - Beispiele: undefinierter OPCODE, Overflow, Syscall,…
  - Interrupt: von externem I/O Controller ausgelöst
- Geschickter Umgang mit Exceptions und Interrupts ohne schwerwiegende Performance-Einbußen ist schwierig

Abarbeitung von Exceptions

- Exceptions müssen vom Betriebssystem (Exception-Handler) behandelt werden
  - Programm-Code ist nicht für Exceptions programmiert
- Exception-Handler steht an vordefinierter Adresse
  - entweder eine Adresse für alle Exceptions
    - in MIPS: 0x8000 0180
  - oder Exception-spezifische Adressen
- Ablauf einer Exception:
  - Speichern des Programm Counters
    - in MIPS: im Register Exception Program Counter (EPC)
  - Speichern der Ursache des Problems
    - In MIPS: Cause Register
    - Beispiele: 0 (undefinierten OPCODE), 1 (ALU-Overflow)
  - Leeren der Pipeline
  - Sprung zu Exception-Handler, der
    - entweder das Problem behebt und das Progamm mit letzter Instruktion wieder aufruft
    - oder das Problem nicht beheben kann und das Programm beendet

Pipeline mit Exceptions: Overflow in EX

IF, ID und EX Flush löschen aktuelle und nachfolgende Instruktion



PC(+4)und Ursache in Registern speichern

Exception-Handler Adresse über MUX in PC

Beispiel für eine Exception

- Exception (Überlauf) bei add in

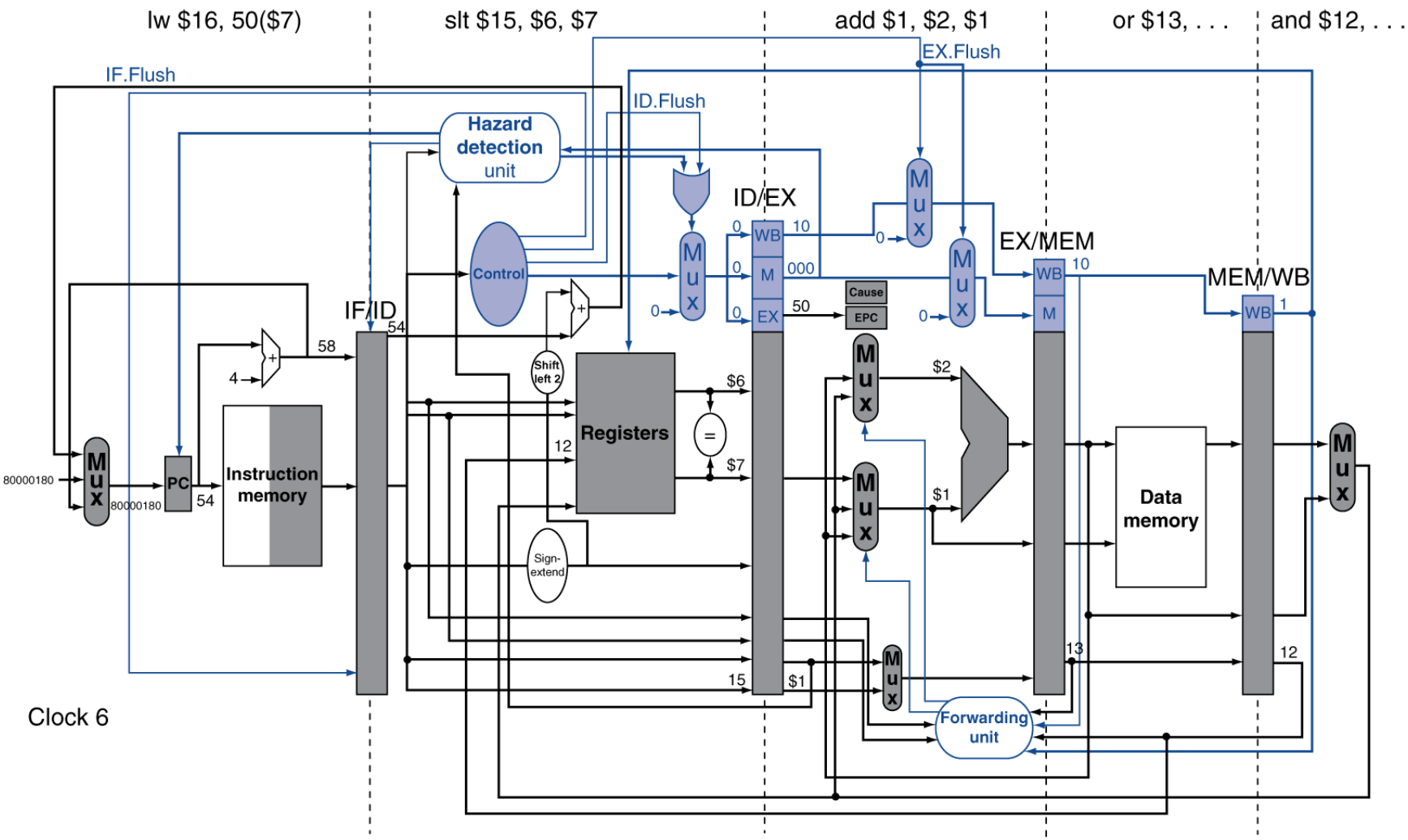
```
40      sub   $11, $2, $4
44      and   $12, $2, $5
48      or    $13, $2, $6
4C      add   $1,  $2, $1
50      slt   $15, $6, $7
54      lw    $16, 50($7)
…
```

- Handler

```
80000180     sw    $26, 1000($0)
80000184     sw    $27, 1004($0)
…
```

  - $26($k0) und $27($k1) sind Register, die der Exception Handler benutzen kann, ohne die Inhalte vor dem Rücksprung wiederherzustellen
  - "normale" Programm sollten diese Register nicht benutzen, da die Werte bei einer Exception überschrieben werden können
  - der Exception Handler stellt alle anderen Register nach einem Rücksprung wieder her

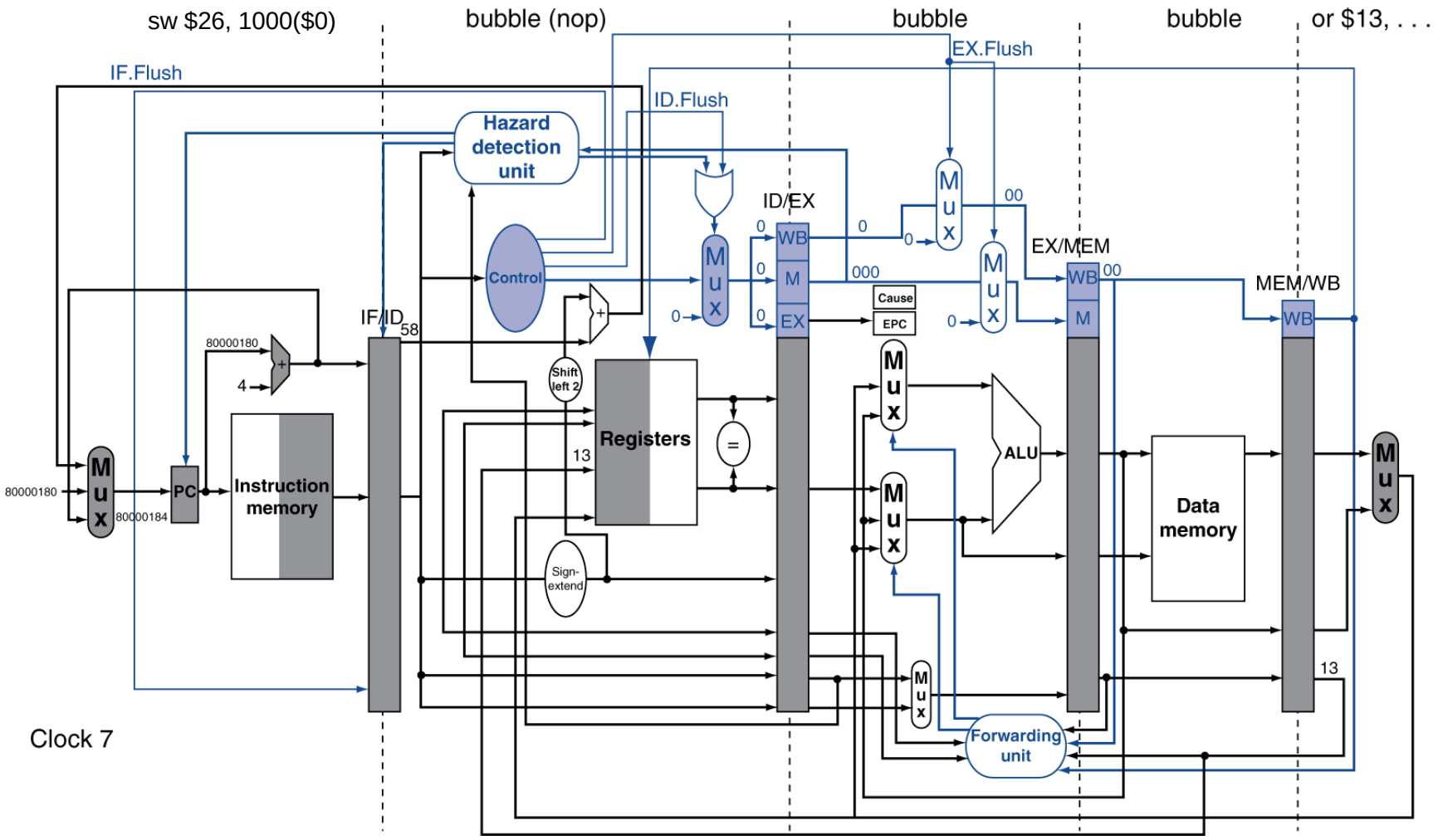
Takt 6: Overflow bei add



Takt 6: Overflow bei add

- die "add" Instruktion steht an Adresse 0x4C (76), so dass in ID/EX-Register.PC der Wert 0x50 (80) steht
- in Takt 6 stellt die ALU einen Overflow fest und setzt ein Overflow-Flag, das eine Exception bewirkt
- das Steuerwerk (Control) bemerkt den Overflow (nicht eingezeichnet) und
  - löscht die IF/ID-, ID/EX- und EX/MEM-Pipeline-Register (Flush)
    - alle Instruktionen nach "add" werden "gelöscht"
    - alle Instruktionen vor "add" werden fertig ausgeführt
  - schreibt die Adresse (0x80000180) in den Program Counter
- PC im ID/EX-Register und Ursache der Exception werden in den Registern EPC und Cause gespeichert
  - in MIPS liegen diese Register im Co-Prozessor 0, der speziell für Exception Handlig da ist

Takt 7: Overflow bei add



Takt 7: Overflow bei add

- in der IF-Stage wird die erste Instruktion des Exception-Handlers geladen, die an Adresse 0x80000180 steht
- in der ID-, EX- und MEM-Stage befinden sich die NOP-Instruktion
- in der WB-Stage wird die "or"-Instruktion fertig ausgeführt
- der PC wird um 4 erhöht und auf 0x800000184 gesetzt, d.h. der Code des Exception Handlers wird jetzt ausgeführt