



Ch.4-5 The Processor - exception

③ Control Hazard 중 Exception

* unexpected events in pipeline

• **exception** CPU 내부에서 발생 (예외처리) ex) overflow

• **interrupt** 외부의 I/O controller에 의해 발생

⇒ 각각의 service control 필요
(각기 다른 ISA는 다 다른 term 사용)

→ 성능에 영향 주지 않고 pipeline operation 변화 주기 어려움...

* **exception** (OS에서는 같다고 취급함. pipeline은 다른 것임 인식)

Interrupts : program 실행과 asynchronous하게 발생

[외부적인 (external) event에 의해 발생

[현재 실행 중인 명령어 모두 실행 → service routine → 다시 시작

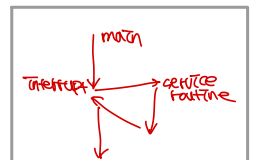
Traps :

" synchronous "

[내부적인 (internal) "

[해당되는 instruction에 대해 trap handler의 도움을 받아서 해결
(pipeline 실행 도중 stop + control 권한도 OS trap handler에게)

→ service routine → 문제였던 명령어의 다음부터만 실행



* Dealing with Exceptions

• exception case

- ▶ R-type arithmetic overflow
- ▶ Trying to execute an undefined instruction
- ▶ An I/O device request
- ▶ An OS service request (e.g., a page fault, TLB exception)
- ▶ A hardware malfunction

• pipeline : 실행 도중 문제가 발생한 instruction 만나면 실행 중지해야 함.

- i) 문제 없는 명령어 complete
- ii) 두(리) 명령어 모두 flush
- iii) register set (exception 경우 나타내도록)
- iv) 문제 생긴 명령어 (offending instruction) address 저장
- v) exception handler code가 있는 곳으로 jump
↳ 이리 defined

⇒ OS : exception 발생 발견 + 해결하는 역할

* exception Handling

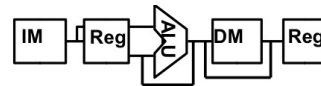
• 언제 발생 → 좀 더 자세히

주어) pipeline 중간에

multiple exceptions
발생가능

- Arithmetic overflow
- Undefined instruction
- TLB or page fault
- I/O service request
- Hardware malfunction

asynchronous
exception
(= interrupt)



Stage(s)?

EX
(ALU 계산 시간)
ID
(decoding 해야 할 수 있음)
IF/MEM
(memory access)

Any

Synchronous?

Y

Y

Y

N

• MIPS에서는 exception 관리 어떻게?

⇒ system control coprocessor (COP)에 의해 관리

offending instruction의 PC 저장 (Exception PC, EPC)

어떠한 문제가 일어났는지, indication 기록

Cause register 원인 field
vector interrupts vector 주소

- ▶ Undefined opcode:
- ▶ Overflow:
- ▶ ...:

원인에 따라 handler로 jump

⇒ handler에 정의된 action 수행

- restartable ① service routine 종료 후 다시 실행
② EPC 이용해서 기존 program으로 return
- o.w ① program 종료
② error report (EPC, cause 이용)

→ 해당 문제에 대한
service routine 정의


C000 0000 = 8000 0000 (hex)
C000 0020 = 8000 0180 (hex)
C000 0040

* 실제 처리과정 살펴보자

① Assume : **EX** 에서 add 수행 → overflow 발생

complete

② add \$1, \$2, \$1 → overflow

2 →  flush :

i) \$1이 영향 받지 않도록 막아야 함

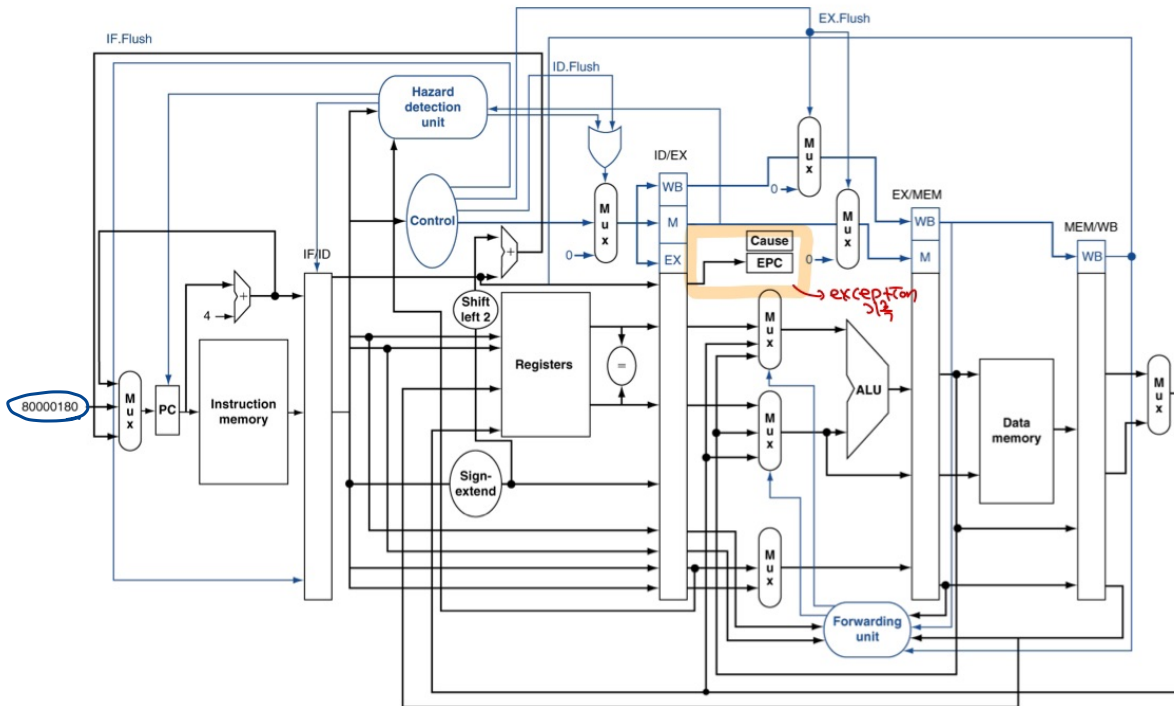
ii) 이전 instruction complete ★

iii) addet add 뒤의 subsequent instruction → flush ★

iv) Cause register EPC reg value setting

v) Handler에 control transfer

⇒ mispredicted branchet 같은 h/w 많지 사용



③ 예외 속성 (exception properties)

• **restartable exception** pipeline은 instruction flush 가능

⇒ handler 수행되고 offending inst를 돌아오면 가능

• PC값 → EPC reg에 저장 (causing instruction 식별)

→ PC+4 값이 저장됨 (다음 instruction 가리킴)

(handler가 무조건부터 조정해야 함)

example) □ Exception on 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)
...
```

→ the overflow occurs

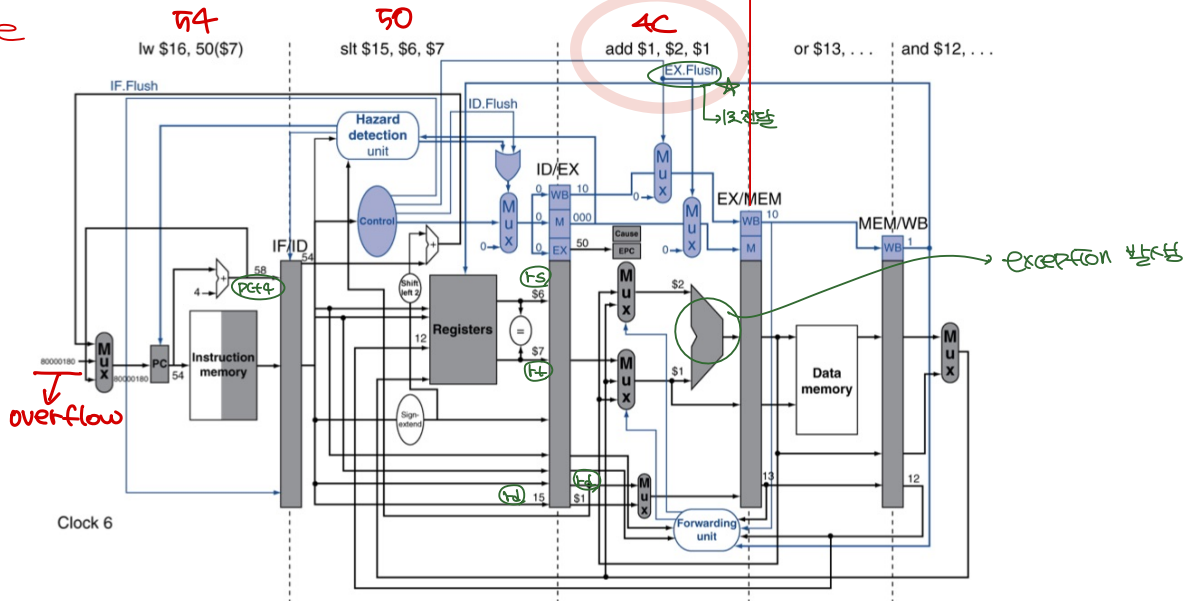
(이제 instruction이 overflow가 됐어
가장빠라!)

□ Handler

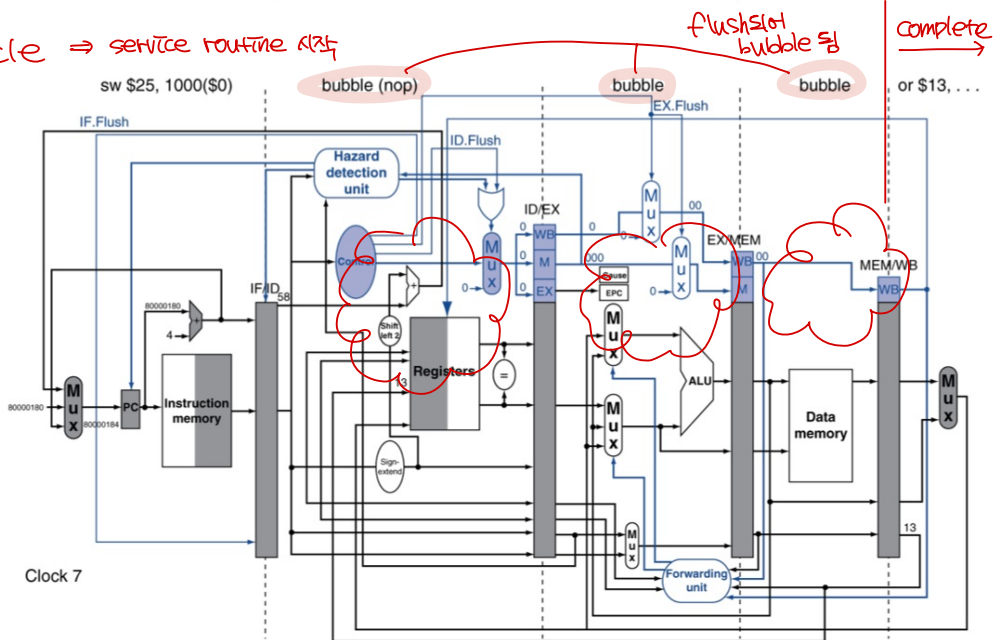
```

80000180  sw  $25, 1000($0)
80000184  sw  $26, 1004($0)
```

a cycle



at1 cycle ⇒ service routine 시작

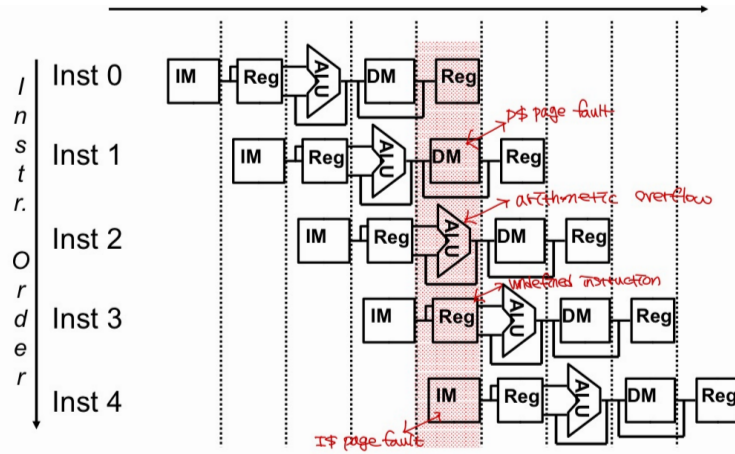


⇒ misprediction과 상충하기위함

⊕ MIPS에 추가해놓은 요소

- CauseWrite : CauseReg control signal
- EPCWrite : EPCReg " "

* Multiple Exceptions



• pipeline \Rightarrow 여러 개의 instruction overlap \rightarrow exception 한 번에 발생 가능

• Solution

i) simple : 가장 먼저 실행하는 inst의 exception 가장 먼저 처리

\Rightarrow 이후 instruction은 flush

\rightarrow 'precise' exceptions : 예외 처리 완료 후 예외 발생 지점에서 다시 프로그램 동작

ii) complex pipelines : 여러 개의 instruction들이 cycle마다 issue가 될 수 있음.

[completion이 복잡함

[처리 define 어려움 \rightarrow precise exception 시도가 어려움

* Imprecise Exceptions

: 예외 처리 도중에 프로그램 종료

• 문제가 된 pipeline stop \rightarrow state 저장 \Rightarrow exception cause 포함

• Handler에게

[어떤 instruction이 exception 가지고 왔는지

[어떤 것들이 complete, flush? \checkmark OS: appending problem 여부 check (manual (수동) completion 필요 가능)

\Rightarrow H/W는 간단 But, handler SW 복잡

\rightarrow 구현 불가능

\Rightarrow precise exception 사용