6.823 SPring15
Quiz 1 Review (L01-L05)

Hsin-Jung Yang

EDSAC

- Accumulator based
- Absolute addressing of memory
  - No index registers
- Use self-modifying code for indirect accesses and subroutine calls

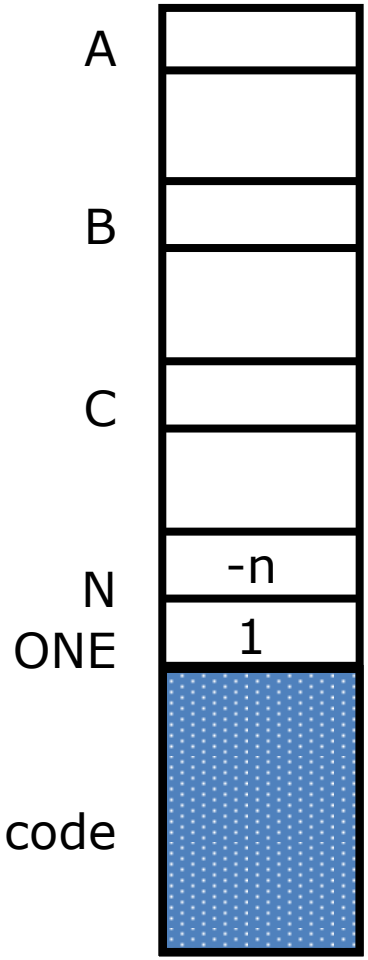
EDSACjr

- A simplified version of EDSAC instruction set

Opcode	Description
ADD *n*	Accum ← Accum + M[*n*]
SUB *n*	Accum ← Accum - M[*n*]
LD *n*	Accum ← M[*n*]
ST *n*	M[*n*] ← Accum
CLEAR	Accum ← 0
OR *n*	Accum ← Accum | M[*n*]
AND *n*	Accum ← Accum & M[*n*]
SHIFTR *n*	Accum ← Accum shiftr *n*
SHIFTL *n*	Accum ← Accum shiftl *n*
BGE *n*	If Accum ≥ 0 then PC ← *n*
BLT *n*	If Accum < 0 then PC ← *n*
END	Halt machine

Programming with EDSACjr

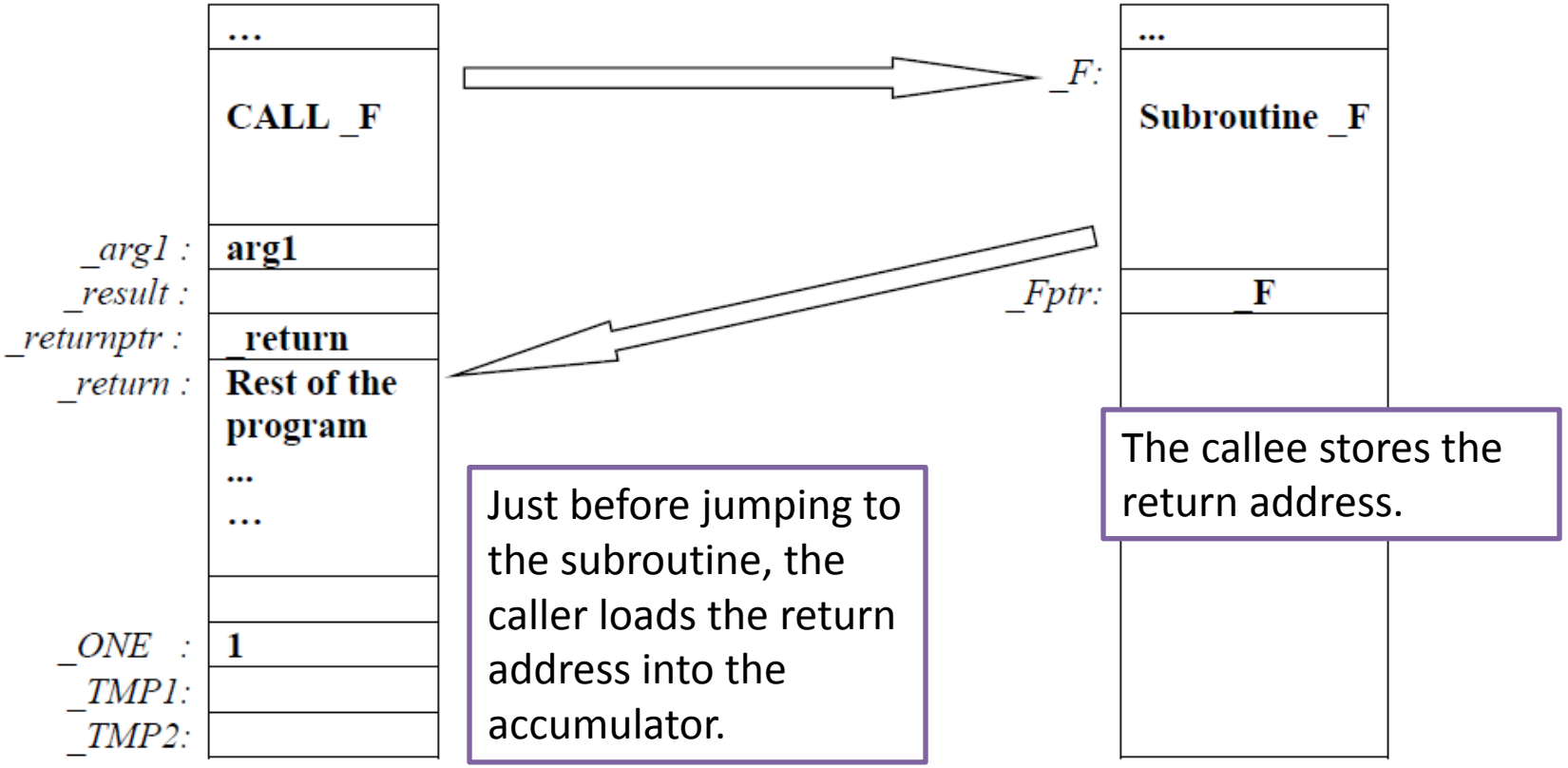
$C_i \leftarrow A_i + B_i, \quad 1 \le i \le n$



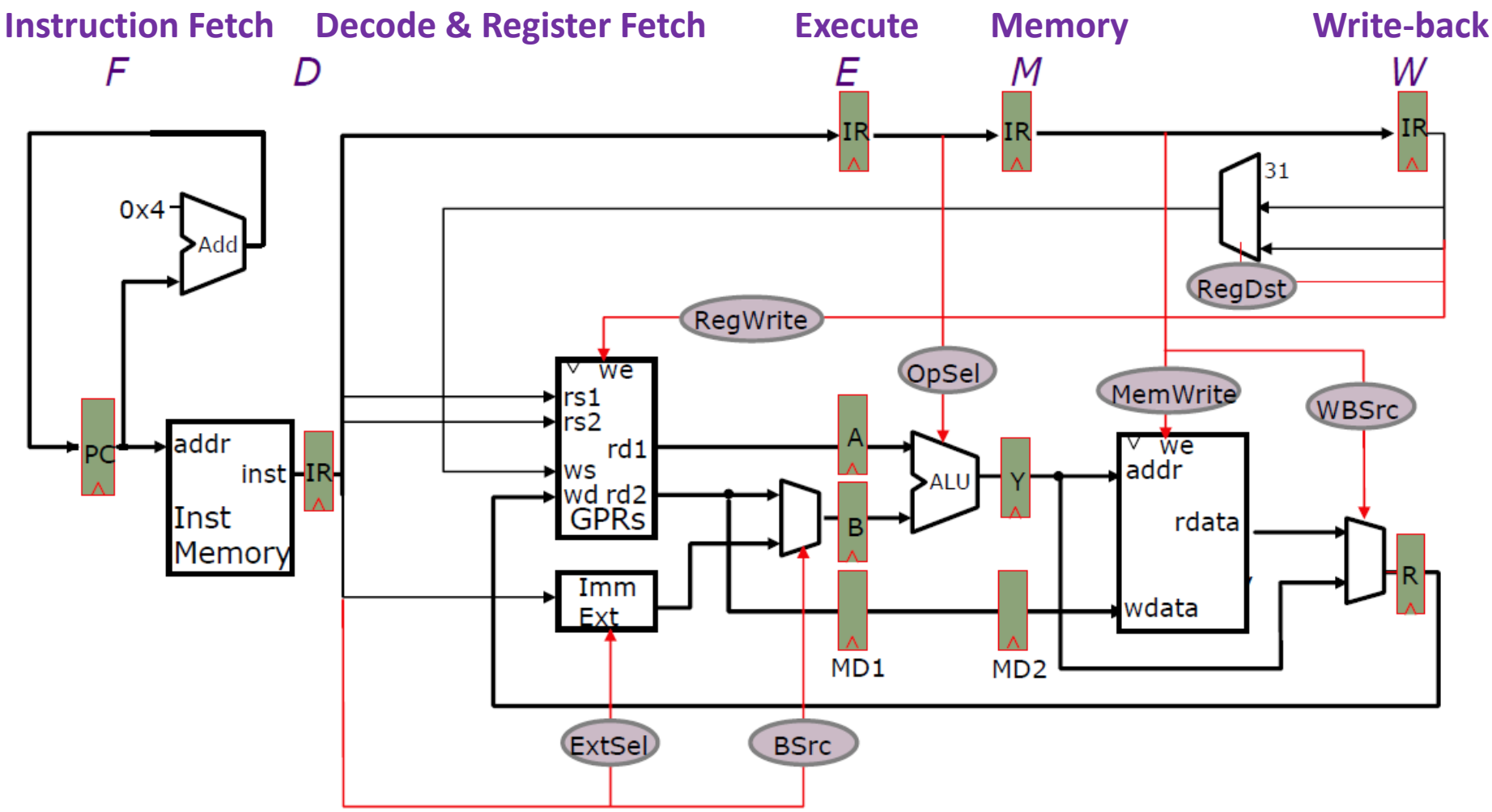

```
LOOP    LD      N
        BGE     DONE
        ADD     ONE
        ST      N
F1      LD      A
F2      ADD     B
F3      ST      C
        LD      F1
        ADD     ONE
        ST      F1
        LD      F2
        ADD     ONE
        ST      F2
        LD      F3
        ADD     ONE
        ST      F3
        CLEAR
        BGE     LOOP
DONE    END
```

Programming with EDSACjr

- Subroutine call in EDSACjr



MIPS 5-Stage Pipeline



Pipeline Hazards

- **Structural Hazard**
  - An instruction in the pipeline may need a resource being used by another instruction in the pipeline
- **Data Hazard**
  - An instruction's read depends on the data produced by an earlier instruction
- **Control Hazard**
  - Next PC calculation depends on the instructions in the pipeline (branches, exceptions)

Pipeline Hazards

- **Structural Hazard**
  - An instruction in the pipeline may need a resource being used by another instruction in the pipeline
  - Ex: Princeton-style architecture
  - Solution: Stall

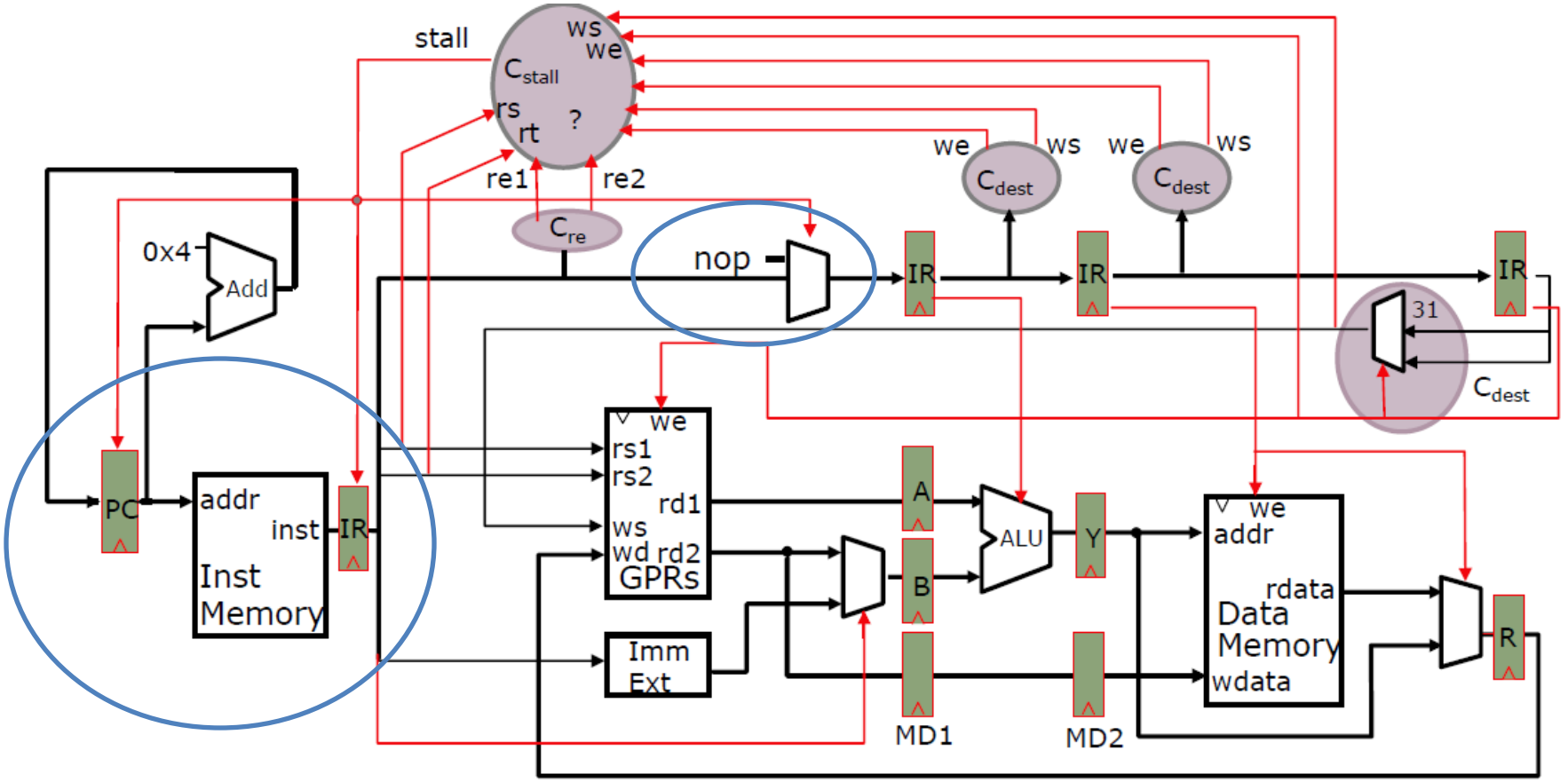
Pipeline Hazards

- **Data Hazard**
  - An instruction's read depends on the data produced by an earlier instruction
  - EX: read-after-write (RAW) data hazard

    r4 <- r1 + 5

    r3 <- r4 + 10
  - Solution: stall or bypass

Resolving Data Hazards By Stalling



Resolving Data Hazards By Stalling

C_{dest}

ws = *Case* opcode

ALU	⇒ rd
ALUi, LW	⇒ rt
JAL, JALR	⇒ R31

we = *Case* opcode

ALU, ALUi, LW	⇒ (ws ≠ 0)
JAL, JALR	⇒ on
...	⇒ off

C_{re}

re1 = *Case* opcode

ALU, ALUi, LW, SW, BZ, JR, JALR	⇒ on
J, JAL	⇒ off

re2 = *Case* opcode

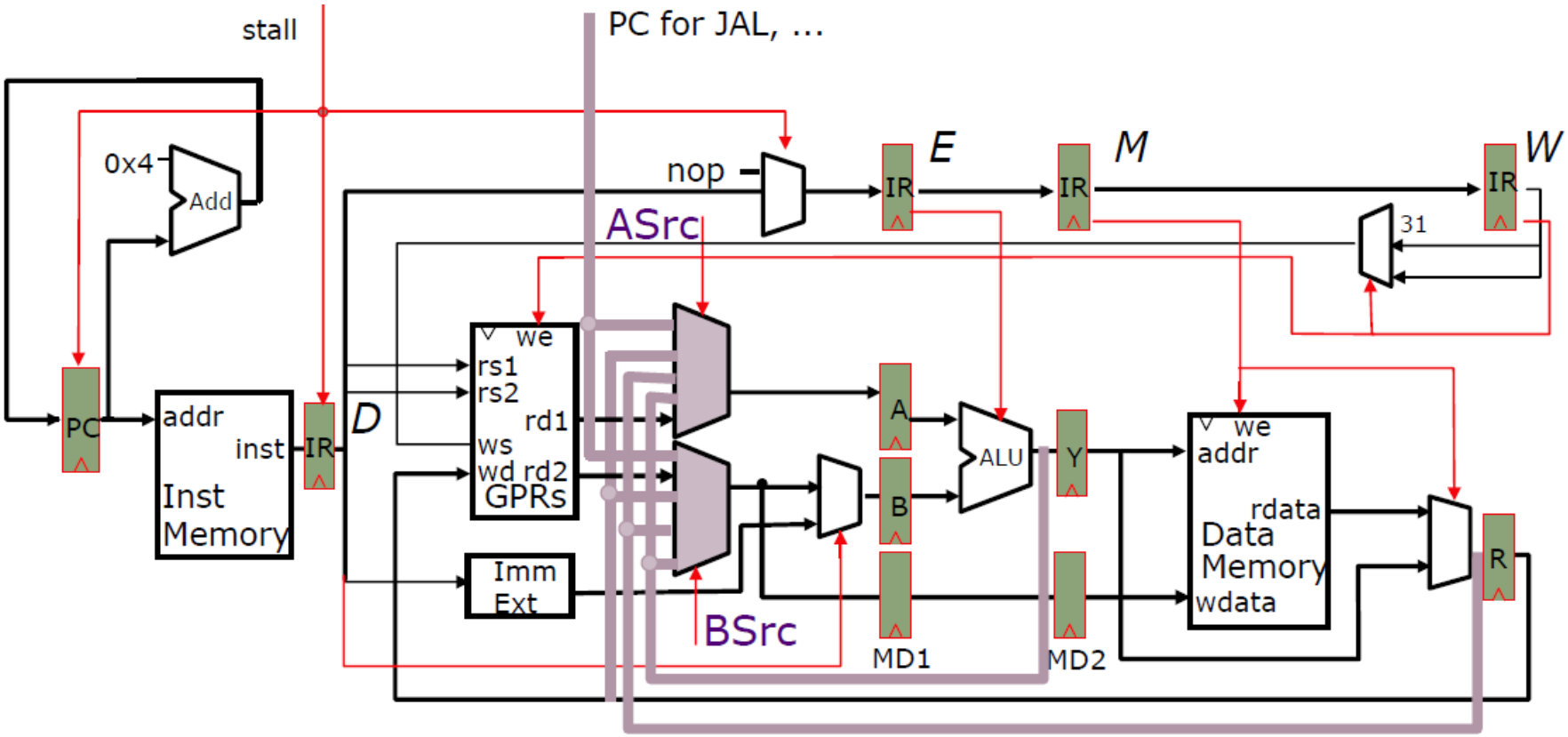
ALU, SW	⇒ on
...	⇒ off

C_{stall}

$$\begin{aligned} stall = ((&rs_D = ws_E)\cdot we_E + \\ &(rs_D = ws_M)\cdot we_M + \\ &(rs_D = ws_W)\cdot we_W)\cdot re1_D \quad + \\ ((&rt_D = ws_E)\cdot we_E + \\ &(rt_D = ws_M)\cdot we_M + \\ &(rt_D = ws_W)\cdot we_W)\cdot re2_D \end{aligned}$$

This is not the full story !

Resolving Data Hazards By Bypassing

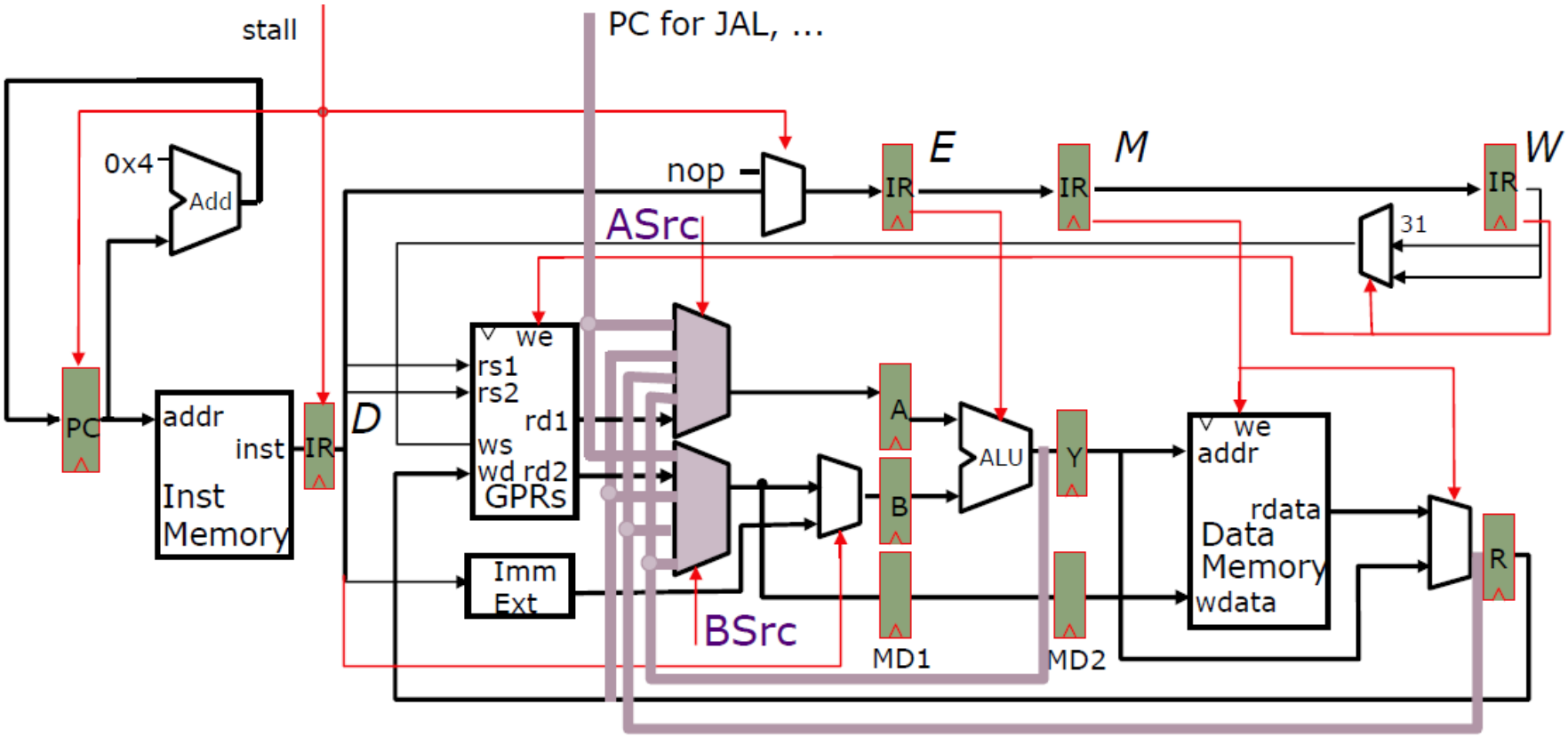


r1 <- M[r2+5]

r3 <- r1 + 10

$$\begin{aligned} stall = {} & (rs_D = ws_E) \cdot (opcode_E = LW_E) \cdot (ws_E \neq 0) \cdot re1_D \\ & + (rt_D = ws_E) \cdot (opcode_E = LW_E) \cdot (ws_E \neq 0) \cdot re2_D \end{aligned}$$

Resolving Data Hazards By Bypassing



Bypass priority: Bypass $_{E->D}$ > Bypass $_{M->D}$ > Bypass $_{W->D}$

Pipeline Hazards

- **Control Hazard**
  - Next PC calculation depends on the instructions in the pipeline (ex: branches, exceptions)
  - Solution1: stall
  - Solution2: speculate

    Guessed correctly → do nothing

    Guessed incorrectly → kill and restart

Quiz 1 Handout (HAL 180)

- **HAL 180**
  - Use condition codes

Name	Description
Sign Flag (SF)	Stores 1 if the result of the *last arithmetic or comparison instruction* was negative, 0 if it was positive
Zero Flag (ZF)	Stores 1 if the result of the *last arithmetic, logical, or comparison instruction* was zero, and 0 if it was non-zero

Table 1. HAL 180 status flags.

Quiz 1 Handout (HAL 180)

- **ISA**

Instruction	Description	SF	ZF
Arithmetic Instructions			
ADD *s1, s2*	*s1* ← *s1* + *s2*	W	W
SUB *s1, s2*	*s1* ← *s1* - *s2*	W	W
MUL *s1, s2*	*s1* ← *s1* × *s2*	W	W
Logical Instructions			
AND *s1, s2*	*s1* ← *s1* & *s2*		W
OR *s1, s2*	*s1* ← *s1* | *s2*		W
XOR *s1, s2*	*s1* ← *s1* ^ *s2*		W
Comparison Instructions			
CMP *s1, s2*	*temp* ← *s1* - *s2*	W	W
Jump Instructions			
JMP *target*	jump to the address specified by *target*		
JL *target*	jump to *target* if SF == 1	R	
JG *target*	jump to *target* if SF == 0 and ZF == 0	R	R
Memory Instructions			
LD *s1, s2*	*s1*← M[*s2*]		
ST *s1, s2*	M[*s1*] ← *s2*		

Table 2. HAL 180 instruction set.

HAL 180: 6-stage Pipeline