

CS 261

Fall 2022

Mike Lam, Professor

Stage	IRMOVQ
Fch	$icode:ifun \leftarrow M_1[PC]$ $rA:rB \leftarrow M_1[PC+1]$ $valC \leftarrow M_8[PC+2]$ $valP \leftarrow PC + 10$
Dec	
Exe	$valE \leftarrow valC$
Mem	
WB	$R[rB] \leftarrow valE$
PC	$PC \leftarrow valP$

Y86 Semantics

Y86 semantics

- **Semantics:** the study of *meaning*
 - What does an instruction "mean"?
 - For us, it is *the effect that it has on the machine*
 - This requires understanding the purpose and actions of all stages of the Y86-64 CPU

Fetch

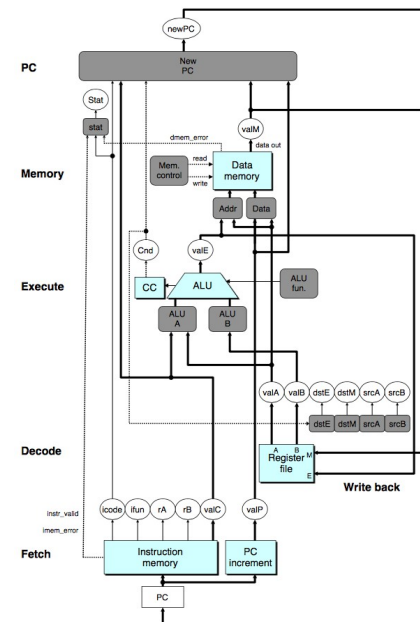
Stage	IRMOVQ
Fch	$icode:ifun \leftarrow M_1[PC]$ $rA:rB \leftarrow M_1[PC+1]$ $valC \leftarrow M_8[PC+2]$ $valP \leftarrow PC + 10$

Decode
Execute

Dec	
Exe	$valE \leftarrow valC$

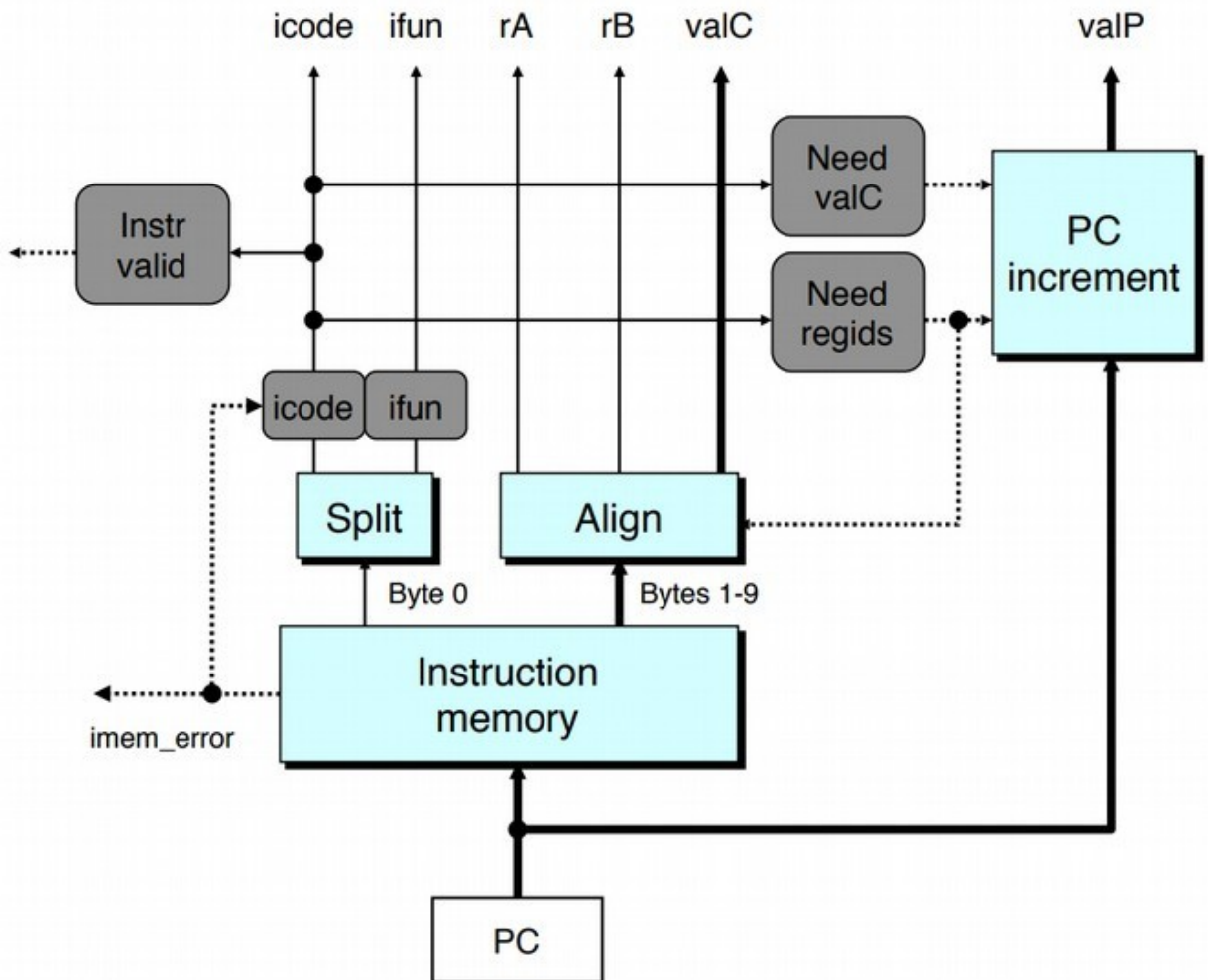
Memory
Write back
PC update

Mem	
WB	$R[rB] \leftarrow valE$
PC	$PC \leftarrow valP$



Fetch

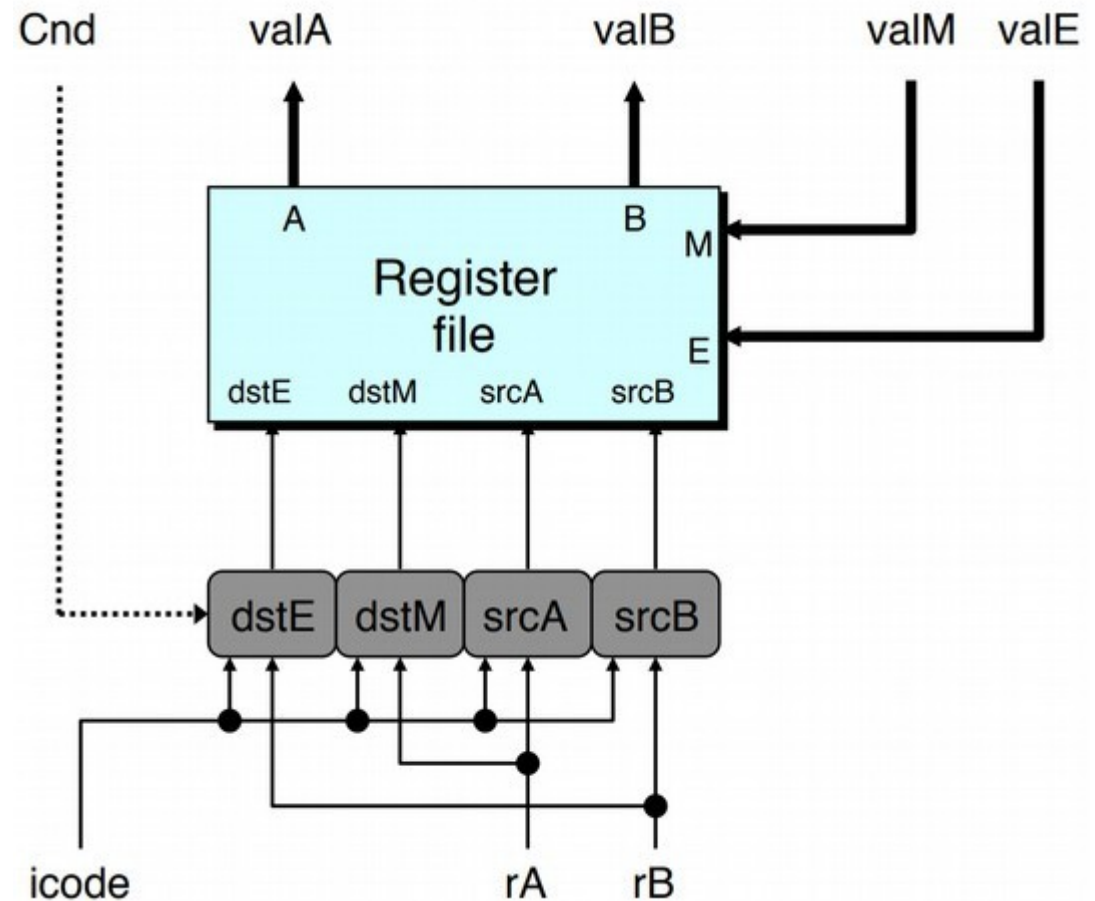
- Read ten bytes from memory at address PC
- Extract instruction fields
 - icode and ifun
 - rA and rB
 - valC
- Compute valP (address of next instruction)
 - $PC + 1 + \text{needsRegIDs} + 8 * \text{needsValC}$



Q: Which instructions have a valC? Which instructions only need the icode and ifun?

Decode

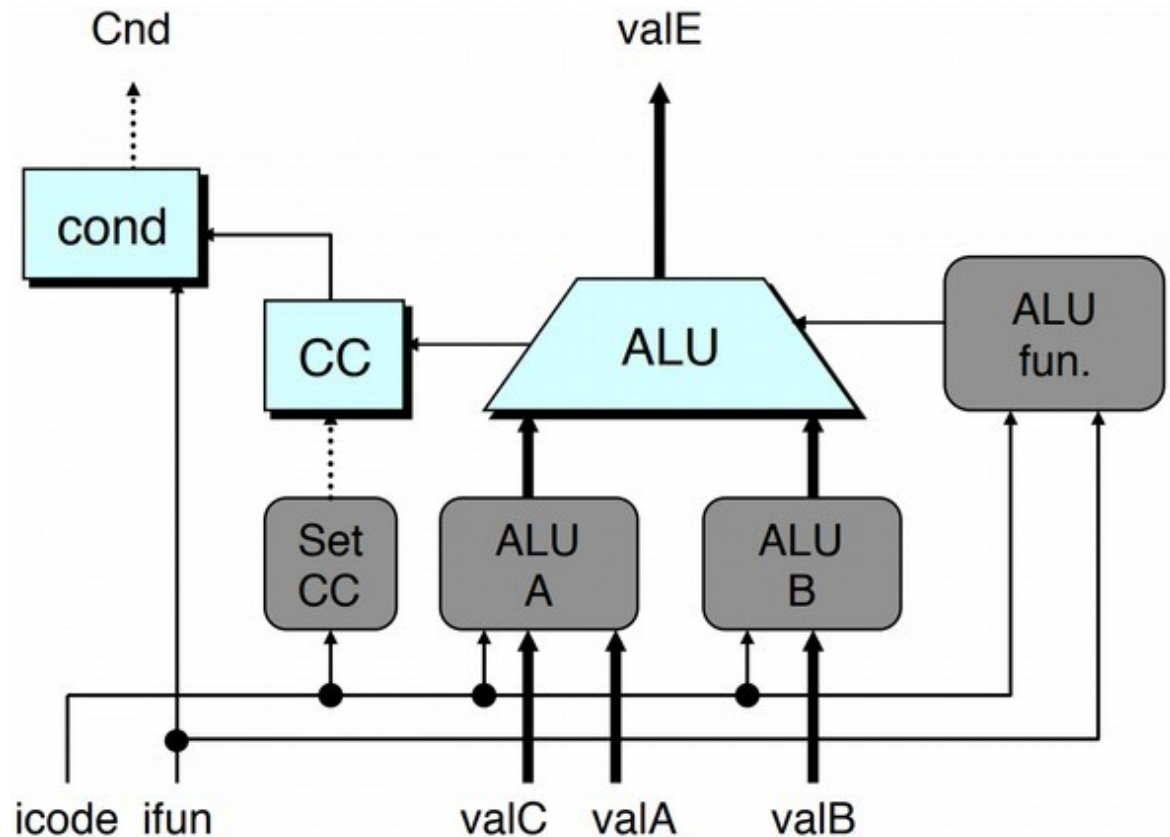
- Read register file
 - Read srcA into valA
 - Read srcB into valB



Q: Which instructions read from both rA and rB?

Execute

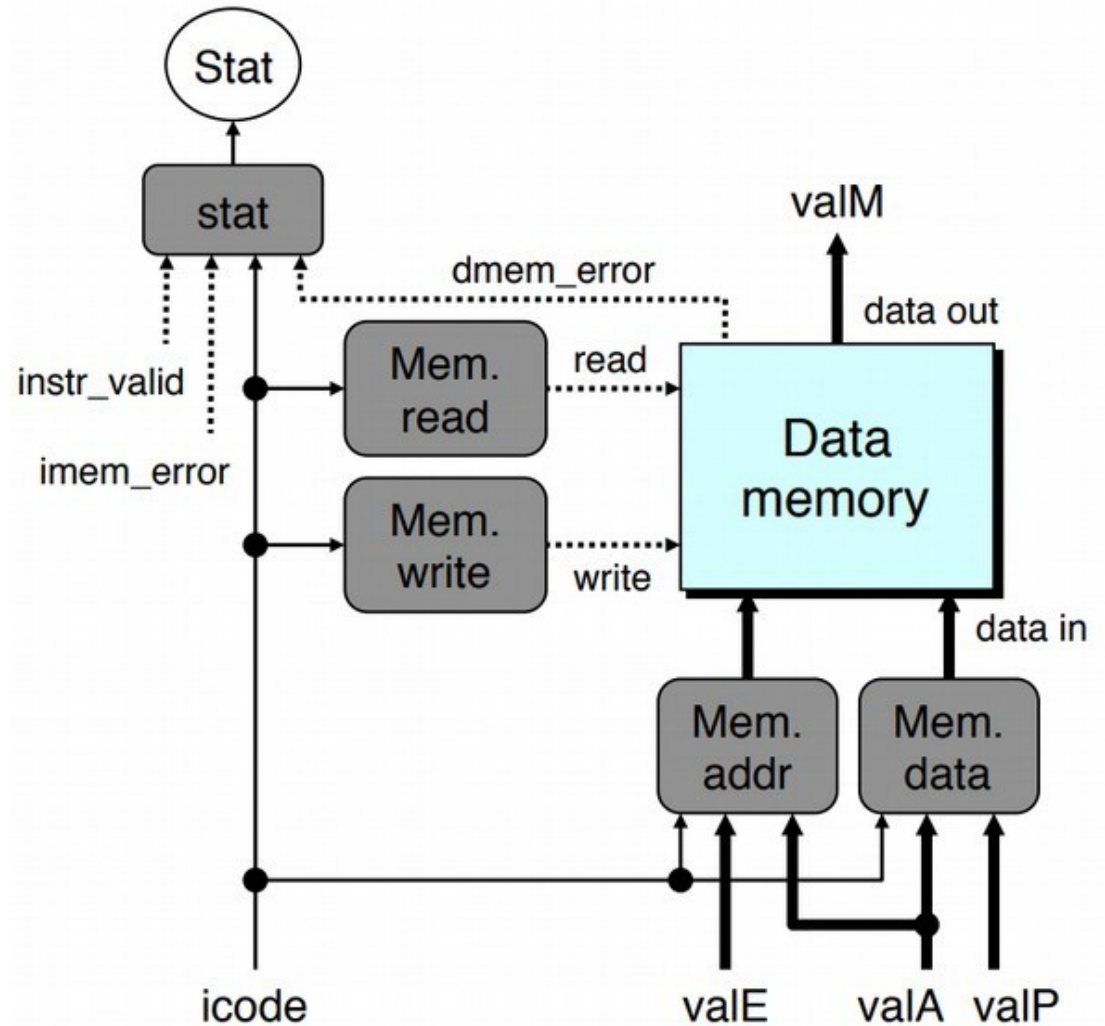
- Perform arithmetic or logic operation
 - Could also be an effective address calculation or stack pointer increment / decrement
 - First input is valC (immediate/offset), valA (register), or a constant (-8 or 8)
 - Second input is valB (register) or zero
- Set condition codes
 - Only if OPq



Q: Which instructions use the ALU?
(Hint: more than you might initially expect!)

Memory

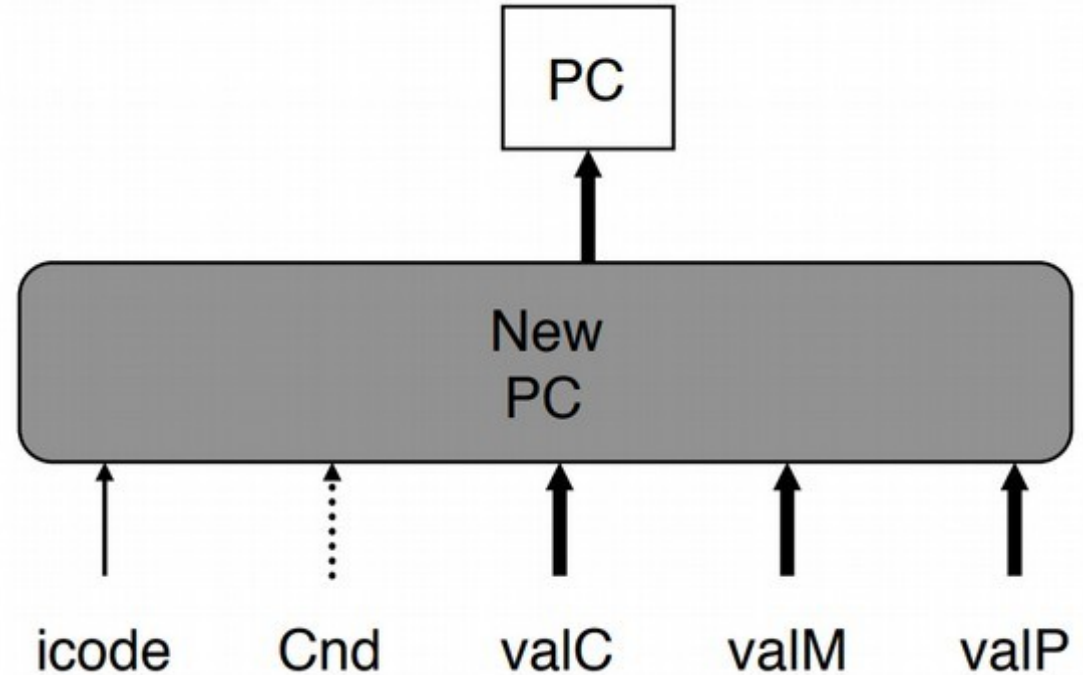
- Read or write memory
 - No instruction does both!
 - Effective address is valE or valA (depending on icode)
 - Data to be written is either valA or valP (depending on icode)
 - Data is read into valM



Q: Which instruction needs to write the address of the next instruction (valP) to memory?

PC update

- Set new PC
 - valP (next instruction) for most icodes
 - Either valP or valC for conditional jumps depending on Cnd
 - valM (return address popped from stack) for ret



Q: Which instruction uses neither valC, valM, or valP to set the PC?

Question

- What effect does the following instruction have?

```
irmovq $128, %rsp
```

- A) It sets RSP to 128
- B) It moves the 64-bit value 128 into memory at the location stored in RSP
- C) It sets RSP to 128 and increments the PC by 10
- D) It pushes the value 128 onto the stack
- E) It pushes the value at address 128 onto the stack

Y86 semantics

- **Semantics**: the study of *meaning*
 - For us, it is *the effect that it has on the machine*
 - We should specify these semantics very **formally**
 - This will help us think correctly about P4
 - ISA reference sheet includes mathematical semantics

In the following semantics, PC and STAT refer to the program counter and status code of the CPU.

Stage	HALT	NOP	cmovXX	IRMOVQ
Fch	icode:ifun $\leftarrow M_1[PC]$ valP $\leftarrow PC + 1$	icode:ifun $\leftarrow M_1[PC]$ valP $\leftarrow PC + 1$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$ valP $\leftarrow PC + 2$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$ valC $\leftarrow M_8[PC+2]$ valP $\leftarrow PC + 10$
Dec			valA $\leftarrow R[rA]$	
Exe	STAT $\leftarrow HLT$		valE $\leftarrow valA$ Cnd $\leftarrow Cond(CC, ifun)$	valE $\leftarrow valC$
Mem				
WB			Cnd ? R[rB] $\leftarrow valE$	R[rB] $\leftarrow valE$
PC	PC $\leftarrow valP$	PC $\leftarrow valP$	PC $\leftarrow valP$	PC $\leftarrow valP$

Aside: syntax notes

- $R[RSP]$ = the value of `%rsp`
- $R[rA]$ = the value of register with id `rA`
- $M_1[PC]$ = the value of one byte in memory at address `PC`
- $M_8[PC+2]$ = the value of eight bytes in memory at address `PC+2`
- $rA:rB = M_1[PC+1]$ means read the byte at address `PC+1`
 - Split it into high- and low-order 4-bits for `rA` and `rB`
- $Cond(CC, ifun)$ returns 0 or 1 based on `CC` and `ifun`
 - Determines whether the given `CMOV/JUMP` should happen
 - See CS:APP 3.6.1-3.6.3 and Figure 3.15
- Convention: write addresses using hex padded to three chars
- Convention: write integer literals using decimal w/ no padding

Example: IRMOVQ

0x016: 30f48000000000000000 |

irmovq \$128,%rsp

Stage	IRMOVQ
Fch	$\text{icode:ifun} \leftarrow M_1[\text{PC}]$ $\text{rA:rB} \leftarrow M_1[\text{PC}+1]$ $\text{valC} \leftarrow M_8[\text{PC}+2]$ $\text{valP} \leftarrow \text{PC} + 10$
Dec	
Exe	$\text{valE} \leftarrow \text{valC}$
Mem	
WB	$R[\text{rB}] \leftarrow \text{valE}$
PC	$\text{PC} \leftarrow \text{valP}$

$\text{icode:ifun} \leftarrow M_1[0x016] = 3:0$
 $\text{rA:rB} \leftarrow M_1[0x017] = f:4$
 $\text{valC} \leftarrow M_8[0x018] = 128$
 $\text{valP} \leftarrow 0x016 + 10 = 0x020$

What effects does this instruction have?

Example: IRMOVQ

0x016: 30f48000000000000000 |

irmovq \$128,%rsp

Stage	IRMOVQ
Fch	$\text{icode:ifun} \leftarrow M_1[\text{PC}]$ $\text{rA:rB} \leftarrow M_1[\text{PC}+1]$ $\text{valC} \leftarrow M_8[\text{PC}+2]$ $\text{valP} \leftarrow \text{PC} + 10$
Dec	
Exe	$\text{valE} \leftarrow \text{valC}$
Mem	
WB	$R[\text{rB}] \leftarrow \text{valE}$
PC	$\text{PC} \leftarrow \text{valP}$

$\text{icode:ifun} \leftarrow M_1[0x016] = 3:0$
 $\text{rA:rB} \leftarrow M_1[0x017] = f:4$
 $\text{valC} \leftarrow M_8[0x018] = 128$
 $\text{valP} \leftarrow 0x016 + 10 = 0x020$

$\text{valE} \leftarrow 128$

$R[\%rsp] \leftarrow \text{valE} = 128$

$\text{PC} \leftarrow \text{valP} = 0x020$

This instruction sets %rsp to 128 and increments the PC by 10

Example: POPQ

0x02c: b00f

$R[\%rsp] = 120$

| popq %rax

$M_8[120] = 9$

Stage	POPQ
Fch	$icode:ifun \leftarrow M_1[PC]$ $rA:rB \leftarrow M_1[PC+1]$
Dec	$valP \leftarrow PC + 2$ $valA \leftarrow R[RSP]$ $valB \leftarrow R[RSP]$
Exe	$valE \leftarrow valB + 8$
Mem	$valM \leftarrow M_8[valA]$
WB	$R[RSP] \leftarrow valE$ $R[rA] \leftarrow valM$
PC	$PC \leftarrow valP$

$icode:ifun \leftarrow M_1[0x02c] = b:0$

$rA:rB \leftarrow M_1[0x02d] = 0:f$

$valP \leftarrow 0x02c + 2 = 0x02e$

$valA \leftarrow R[\%rsp] = 120$

$valB \leftarrow R[\%rsp] = 120$

$valE \leftarrow 120 + 8 = 128$

$valM \leftarrow M_8[120] = 9$

$R[\%rsp] \leftarrow 128$

$R[\%rax] \leftarrow 9$

$PC \leftarrow 0x02e$

This instruction sets %rax to 9, sets %rsp to 128, and increments the PC by 2

Example: CALL

0x037: 80410000000000000000 | call proc

R[%rsp] = 128

Stage	CALL	
Fch	$\text{icode:ifun} \leftarrow M_1[\text{PC}]$	$\text{icode:ifun} \leftarrow M_1[0x037] = 8:0$
	$\text{valC} \leftarrow M_8[\text{PC}+1]$	$\text{valC} \leftarrow M_8[0x038] = 0x041$
	$\text{valP} \leftarrow \text{PC} + 9$	$\text{valP} \leftarrow 0x037 + 9 = 0x040$
Dec		
	$\text{valB} \leftarrow R[\text{RSP}]$	$\text{valB} \leftarrow R[\%rsp] = 128$
Exe	$\text{valE} \leftarrow \text{valB} - 8$	$\text{valE} \leftarrow 128 - 8 = 120$
Mem	$M_8[\text{valE}] \leftarrow \text{valP}$	$M_8[120] \leftarrow 0x040$
WB	$R[\text{RSP}] \leftarrow \text{valE}$	$R[\%rsp] \leftarrow 120$
PC	$\text{PC} \leftarrow \text{valC}$	$\text{PC} \leftarrow 0x041$

This instruction sets %rsp to 120, stores the return address 0x040 at [%rsp], and sets the PC to 0x041

Y86 semantics

In the following semantics, PC and STAT refer to the program counter and status code of the CPU.

Stage	HALT	NOP	cmovXX	IRMOVQ
Fch	icode:ifun $\leftarrow M_1[PC]$	icode:ifun $\leftarrow M_1[PC]$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$ valC $\leftarrow M_8[PC+2]$
Dec	valP $\leftarrow PC + 1$	valP $\leftarrow PC + 1$	valP $\leftarrow PC + 2$ valA $\leftarrow R[rA]$	valP $\leftarrow PC + 10$
Exe	STAT $\leftarrow HLT$		valE $\leftarrow valA$ Cnd $\leftarrow Cond(CC,ifun)$	valE $\leftarrow valC$
Mem				
WB			Cnd ? R[rB] $\leftarrow valE$	R[rB] $\leftarrow valE$
PC	PC $\leftarrow valP$	PC $\leftarrow valP$	PC $\leftarrow valP$	PC $\leftarrow valP$
Stage	RMMOVQ	MRMOVQ	OPq	jXX
Fch	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$ valC $\leftarrow M_8[PC+2]$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$ valC $\leftarrow M_8[PC+2]$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$	icode:ifun $\leftarrow M_1[PC]$ valC $\leftarrow M_8[PC+1]$
Dec	valP $\leftarrow PC + 10$	valP $\leftarrow PC + 10$	valP $\leftarrow PC + 2$ valA $\leftarrow R[rA]$	valP $\leftarrow PC + 9$
Exe	valA $\leftarrow R[rA]$ valB $\leftarrow R[rB]$ valE $\leftarrow valB + valC$	valB $\leftarrow R[rB]$ valE $\leftarrow valB + valC$	valB $\leftarrow R[rB]$ valE $\leftarrow valB OP valA$ Set CC	Cnd $\leftarrow Cond(CC,ifun)$
Mem	M ₈ [valE] $\leftarrow valA$	valM $\leftarrow M_8[valE]$		
WB		R[rA] $\leftarrow valM$	R[rB] $\leftarrow valE$	
PC	PC $\leftarrow valP$	PC $\leftarrow valP$	PC $\leftarrow valP$	PC $\leftarrow Cnd ? valC:valP$
Stage	CALL	RET	PUSHQ	POPQ
Fch	icode:ifun $\leftarrow M_1[PC]$	icode:ifun $\leftarrow M_1[PC]$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$	icode:ifun $\leftarrow M_1[PC]$ rA:rB $\leftarrow M_1[PC+1]$
Dec	valC $\leftarrow M_8[PC+1]$ valP $\leftarrow PC + 9$	valP $\leftarrow PC + 1$ valA $\leftarrow R[RSP]$	valP $\leftarrow PC + 2$ valA $\leftarrow R[rA]$	valP $\leftarrow PC + 2$ valA $\leftarrow R[RSP]$
Exe	valB $\leftarrow R[RSP]$ valE $\leftarrow valB - 8$	valB $\leftarrow R[RSP]$ valE $\leftarrow valB + 8$	valB $\leftarrow R[RSP]$ valE $\leftarrow valB - 8$	valB $\leftarrow R[RSP]$ valE $\leftarrow valB + 8$
Mem	M ₈ [valE] $\leftarrow valP$	valM $\leftarrow M_8[valA]$	M ₈ [valE] $\leftarrow valA$	valM $\leftarrow M_8[valA]$
WB	R[RSP] $\leftarrow valE$	R[RSP] $\leftarrow valE$	R[RSP] $\leftarrow valE$	R[RSP] $\leftarrow valE$ R[rA] $\leftarrow valM$
PC	PC $\leftarrow valC$	PC $\leftarrow valM$	PC $\leftarrow valP$	PC $\leftarrow valP$

Y86 CPU (P4)

von Neumann architecture

- 1) Fetch ← P3!
 - Splits instruction at PC into pieces
 - Save info in `y86_inst_t` struct
- 2) Decode (register file)
 - Reads registers
 - P4: Sets `valA`
- 3) Execute (ALU)
 - Arithmetic/logic operation, effective address calculation, or stack pointer increment/decrement
 - P4: Sets `Cnd` and returns `valE`
- 4) Memory (RAM)
 - Reads/writes memory
- 5) Write back (register file)
 - Sets registers
- 6) PC update
 - Sets new PC

