CS 261
Fall 2016

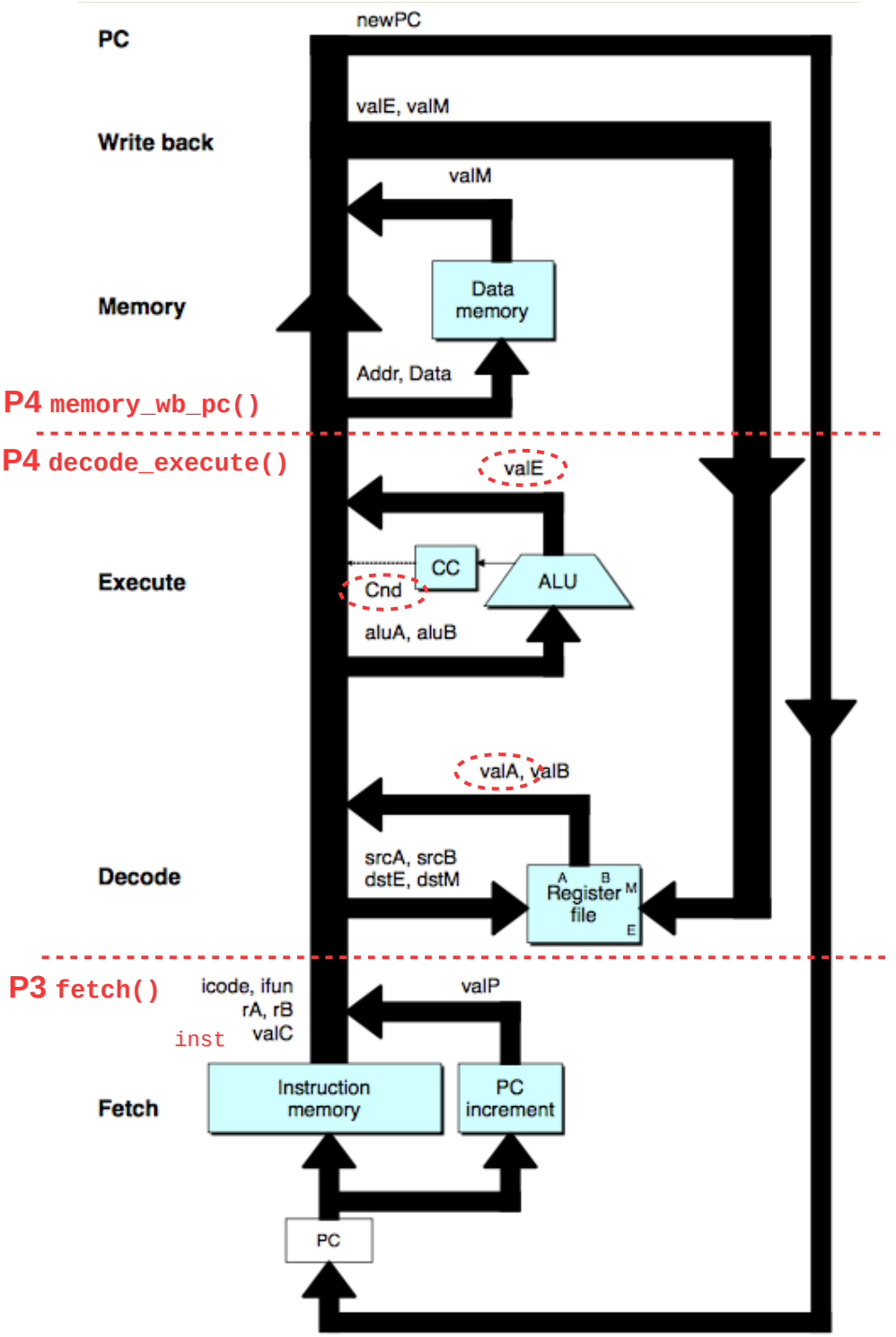
Mike Lam, Professor

Y86 Instruction Semantics

CPU stages

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 `valE` and `Cnd`
4) Memory (RAM)
   - Reads/writes memory
5) Write back (register file)
   - Sets registers
6) PC update
   - Sets new PC



Y86 semantics

- Semantics: the study of meaning
  - What does an instruction "mean"?
  - For us, it means the effect that it has on the machine
  - We should specify these semantics very formally
  - This will help us think correctly about P4

Stage	HALT	NOP	CMOV	IRMOVQ
Fch	icode ← M_1[PC]	icode ← M_1[PC]	icode:ifun ← M_1[PC]	icode:ifun ← M_1[PC]
			rA:rB ← M_1[PC+1]	rA:rB ← M_1[PC+1]
				valC ← M_8[PC+2]
	valP ← PC + 1	valP ← PC + 1	valP ← PC + 2	valP ← PC + 10
Dec			valA ← R[rA]	
Exe	cpu.stat = HLT		valE ← valA	valE ← valC
			Cnd ← Cond(CC,ifun)	
Mem				
WB			Cnd ? R[rB] ← valE	R[rB] ← valE
PC	PC ← 0	PC ← valP	PC ← valP	PC ← 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
- Convention: write addresses using hex padded to three chars
- Convention: write integer literals using decimal w/ no padding

Y86 semantics

Stage	HALT	NOP	CMOV	IRMOVQ
Fch	icode ← M_1[PC] valP ← PC + 1	icode ← M_1[PC] valP ← PC + 1	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC + 10
Dec			valA ← R[rA]	
Exe	cpu.stat = HLT		valE ← valA Cnd ← Cond(CC,ifun)	valE ← valC
Mem				
WB			Cnd ? R[rB]← valE	R[rB] ← valE
PC	PC ← 0	PC ← valP	PC ← valP	PC ← valP

Stage	RMMOVQ	MRMOVQ	OPq	JUMP
Fch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC + 10	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC + 10	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC + 9
Dec	valA ← R[rA] valB ← R[rB]	 valB ← R[rB]	valA ← R[rA] valB ← R[rB]	
Exe	valE ← valB + valC	valE ← valB + valC	valE ← valB OP valA	Cnd ← Cond(CC,ifun)
Mem	M_8[valE] ← valA	valM ← M_8[valE]		
WB		R[rA] ← valM	R[rB] ← valE	
PC	PC ← valP	PC ← valP	PC ← valP	PC ← Cnd?valC:valP

Stage	CALL	RET	PUSHQ	POPQ
Fch	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC + 9	icode:ifun ← M_1[PC] valP ← PC + 1	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2
Dec	 valB ← R[RSP]	valA ← R[RSP] valB ← R[RSP]	valA ← R[rA] valB ← R[RSP]	valA ← R[RSP] valB ← R[RSP]
Exe	valE ← valB - 8	valE ← valB + 8	valE ← valB - 8	valE ← valB + 8
Mem	M_8[valE] ← valP	valM ← M_8[valA]	M_8[valE] ← valA	valM ← M_8[valA]
WB	R[RSP] ← valE	R[RSP] ← valE	R[RSP] ← valE	R[RSP] ← valE R[rA] ← valM
PC	PC ← valC	PC ← valM	PC ← valP	PC ← valP

Example: IRMOVQ

```
0x016: 30f48000000000000000 |     irmovq $128,%rsp
```

Stage	IRMOVQ	
Fch	icode:ifun ← M_1[PC]	icode:ifun ← M_1[0x016] = 3:0
	rA:rB ← M_1[PC+1]	rA:rB ← M_1[0x017] = f:4
	valC ← M_8[PC+2]	valC ← M_8[0x018] = 128
	valP ← PC + 10	valP ← 0x016 + 10 = 0x020
Dec		
Exe	valE ← valC	valE ← 128
Mem		
WB	R[rB] ← valE	R[%rsp] ← valE = 128
PC	PC ← valP	PC ← valP = 0x020

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

Example: POPQ

```
0x02c: b00f                |     popq  %rax
```

R[%rsp] = 120 M_8[120] = 9

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

icode:ifun ← M_1[0x02c] = b:0
rA:rB ← M_1[0x02d] = 0:f

valP ← 0x02c + 2 = 0x02e

valA ← R[%rsp] = 120
valB ← R[%rsp] = 120

valE ← 120 + 8 = 128

valM ← M_8[120] = 9

R[%rsp] ← 128
R[%rax] ← 9

PC ← 0x02e

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

Example: CALL

```
0x037: 8041000000000000    |     call proc
```

R[%rsp] = 128

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

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