Architecture Lab Recitation

Fall 2023

Scenario:

- Implementing basic array algorithms, jump tables
- Implementing indirect jump instructions

Goals:

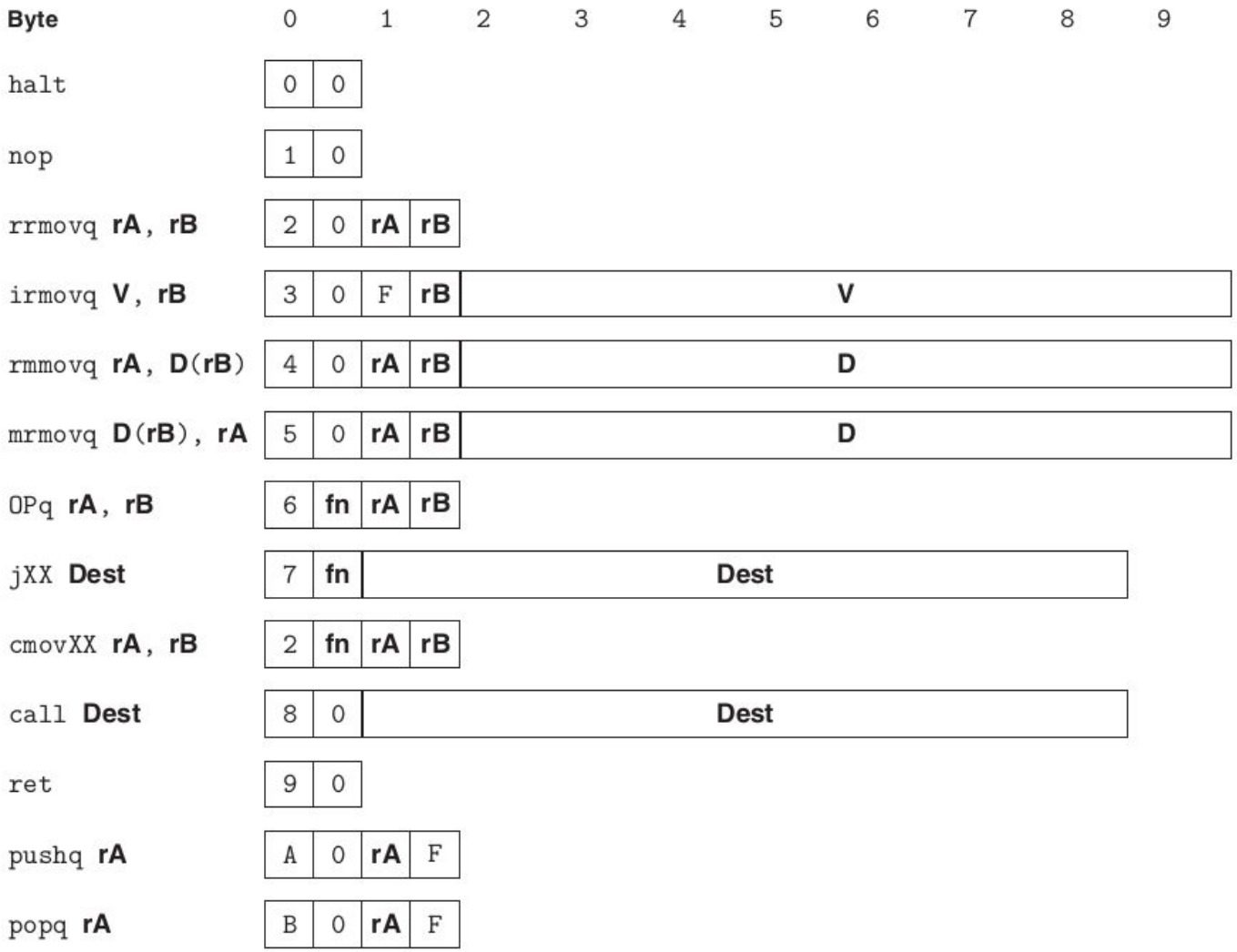
- Mastering efficient assembly, writing full programs
- Understanding details of pipelining
- Speeding up procedures
- Grasping hazards

Tasks

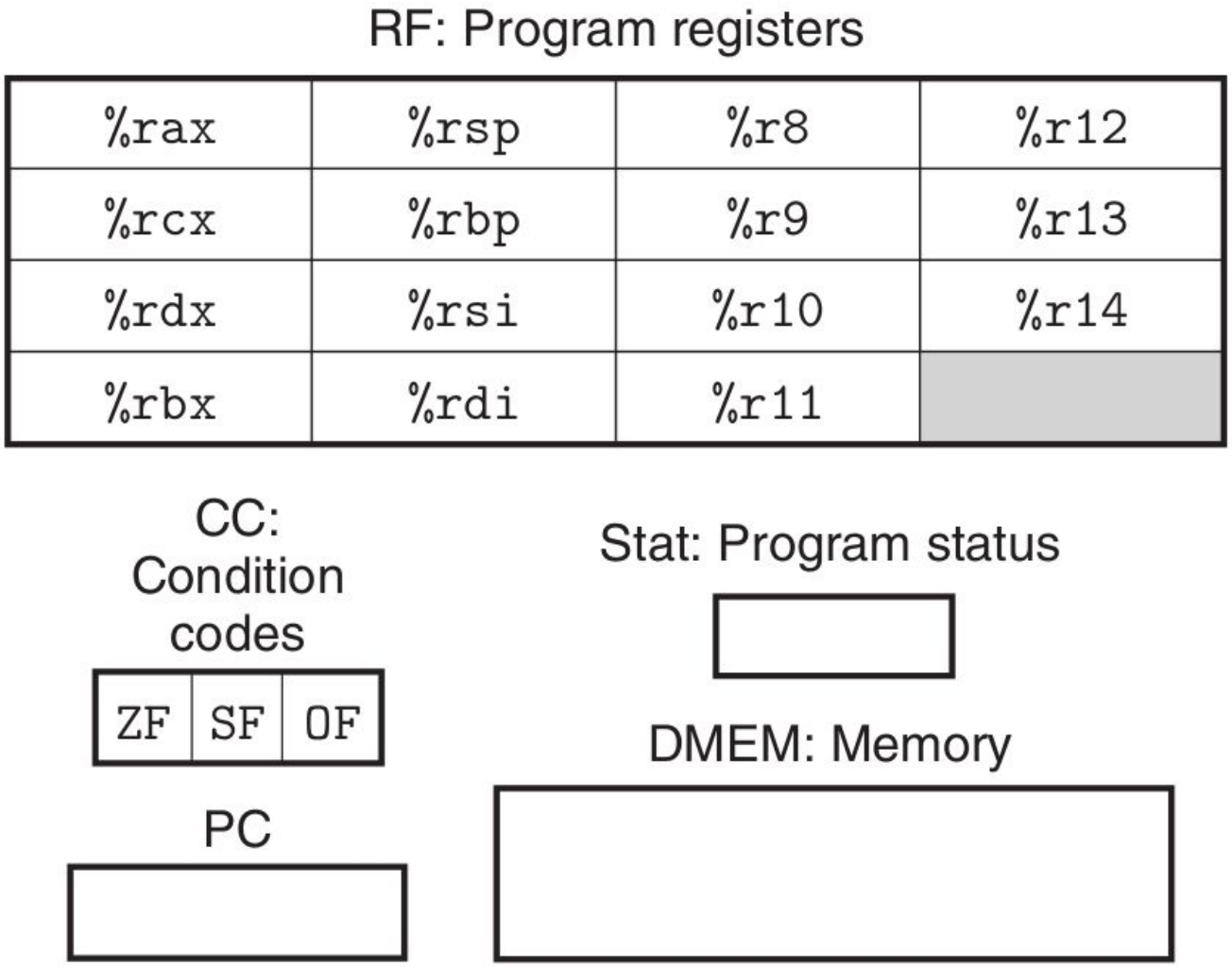
- Part A: Implement Assembly Programs
  - Binary Search
  - Merge
- Part B: Implement `jtab` in SEQ
- Part C: Implement `jtab` in PIPE, and speed up a switch statement.

Y86 Overview

Extra instructions: mllq dvvq modq



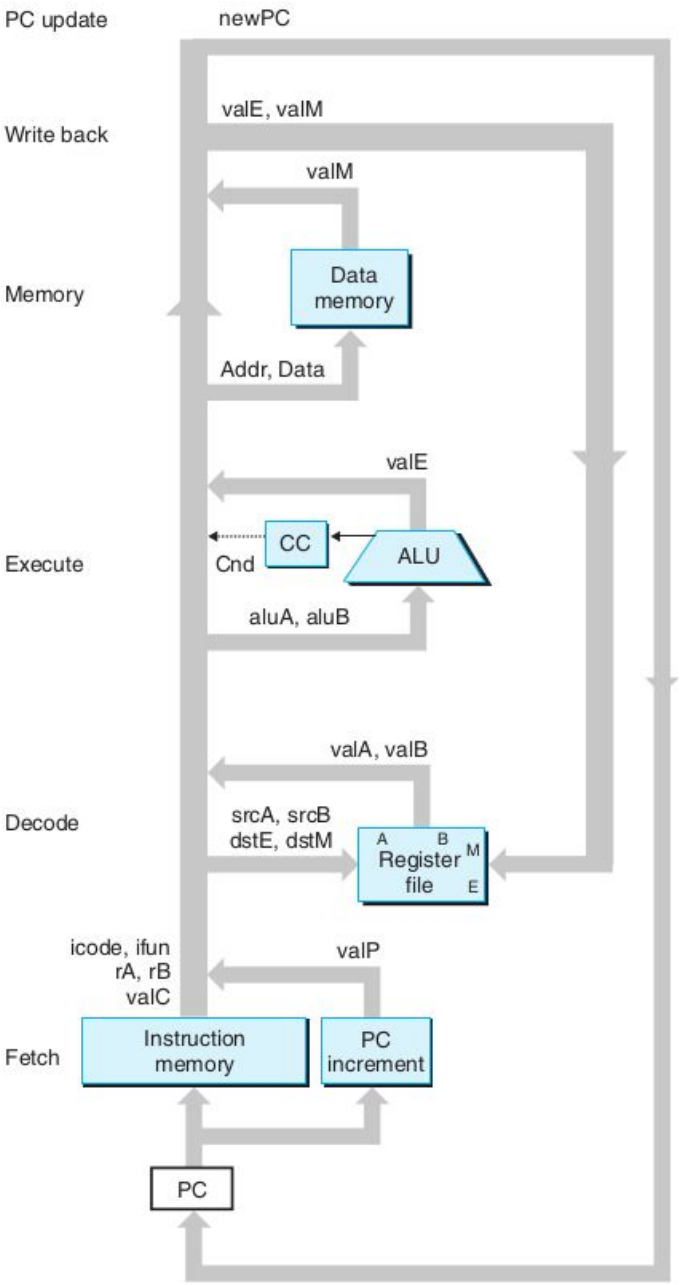
Y86 Overview



Part A Quickstart

- Make the whole handout first.
- Within `misc`:
  - Compile with `yas`
  - Simulate with `yis`
- Within `verifiers`:
  - `python3 binary_search_verifier.py yis <bs.ys path>`
  - `python3 merge_verifier.py yis <merge.ys path>`
- Verifier scripts compile, execute and verify solution in one place
- Tinker with scripts to test/debug more

SEQ


PC update
newPC
valE, valM
Write back
valM
Memory
Data memory
Addr, Data
valE
CC
Cnd
ALU
Execute
aluA, aluB
valA, valB
srcA, srcB
dstE, dstM
A
B
M
Register file
E
Decode
icode, ifun
rA, rB
valC
valP
Instruction memory
PC increment
Fetch
PC

`jtab` instruction

- Indirect Jump (similar to `jmp*` in x86)

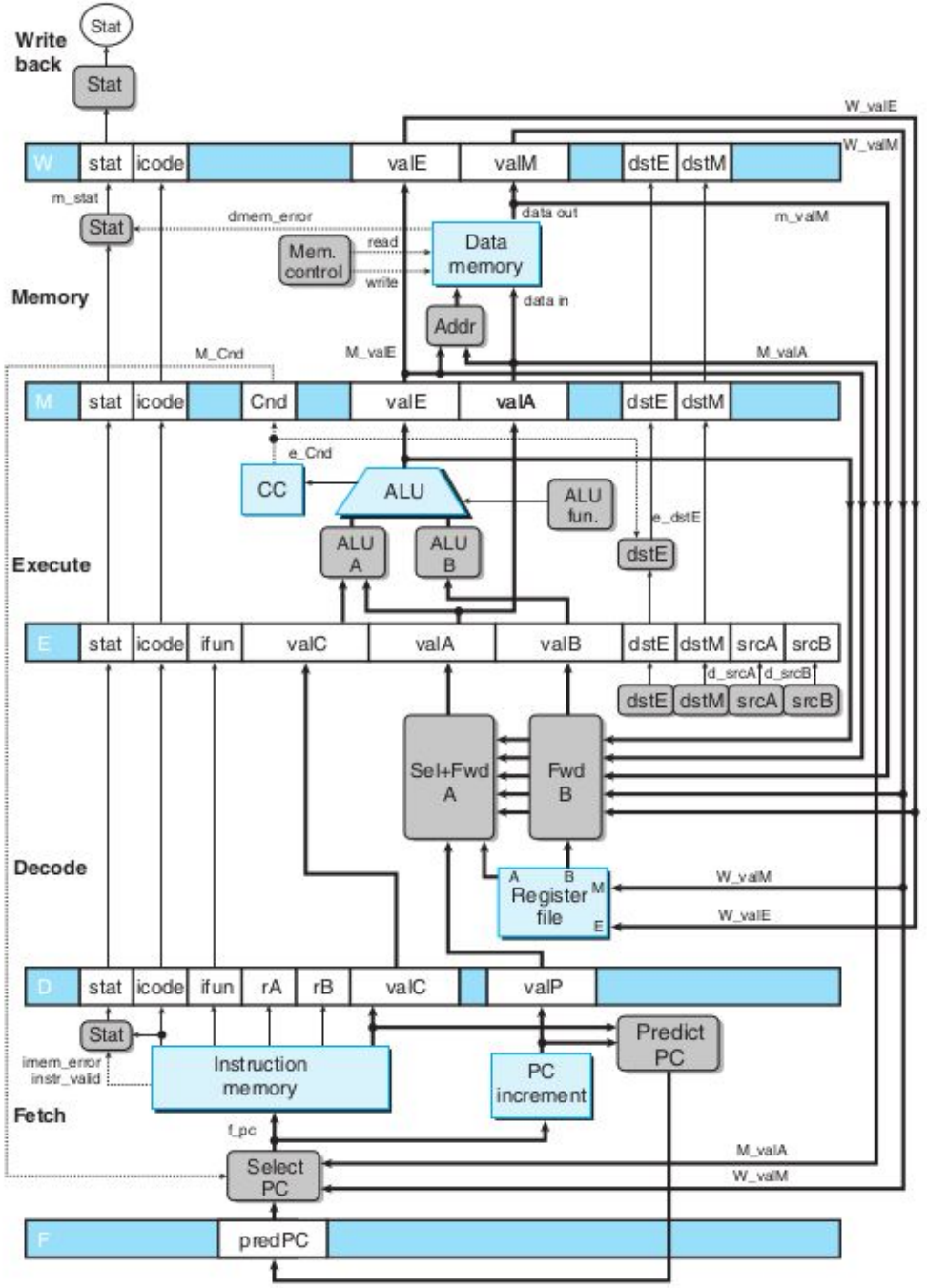
```
jtab C, %rB
```

D	0	F	rB	Constant

Part B Quickstart

- Modify seq-full.hcl, then make
- Two “official” tests
  - Within `y86_code`: `make testssim`
  - Within `ptests`: `make SIM=../seq/ssim`
- Test `jtab` on your own:
  - `jtabtest*.yo` within `sample_files`
  - Your own `switch8` implementation

PIPE


Write back
Stat
Stat
W_valE
W_valM
W
stat
icode
valE
valM
dstE
dstM
m_stat
Stat
dmem_error
data out
m_valM
Mem. control
read
write
Data memory
Memory
data in
Addr
M_Cnd
M_valE
M_valA
M
stat
icode
Cnd
valE
valA
dstE
dstM
e_Cnd
CC
ALU
ALU fun.
e_dstE
ALU A
ALU B
dstE
Execute
E
stat
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
d_srcB
dstE
dstM
srcA
srcB
Sel+Fwd A
Fwd B
Decode
A
B
M
Register file
E
W_valM
W_valE
D
stat
icode
ifun
rA
rB
valC
valP
Stat
Predict PC
imem_error
instr_valid
Instruction memory
PC increment
Fetch
f_pc
M_valA
W_valM
Select PC
F
predPC

Part C Quickstart

- Implement `switch8` and test it with the verifier (no need to implement `jtab`)
  - Can you make it faster without `jtab`?
- Implement `jtab`
- Test with y86_code and ptests:
  - Within `y86_code`: `make testssim`
  - Within `ptests`: `make SIM=../seq/ssim`
- Increase CPE as much as you can in:
  - `python3 switch8_verifier.py psim switch8.ys`

Final Advice

- Find a peaceful weekend day and read the chapter very carefully, taking notes. Make sure your understanding of the architecture is excellent.
- Now, what kind of hazards can `jtab` have?
- Does it share these hazards with other instructions? Which ones? How are the hazards handled for them? How can these be adapted to `jtab`?

Demo