OUTLINE

Section 1. Programming (90%)

- Pipelined CPU (74%)
- Report (16%)

-

Section 2. Submission

- Submission format
- Late submission rules

-

Section 3. Deadline

- Deadline: 2022/12/25 23:59

-

Section 4. Supplementary

- Homework introduction video

-

Programming (90%, including: Pipelined CPU and report)

Pipelined CPU (74%)

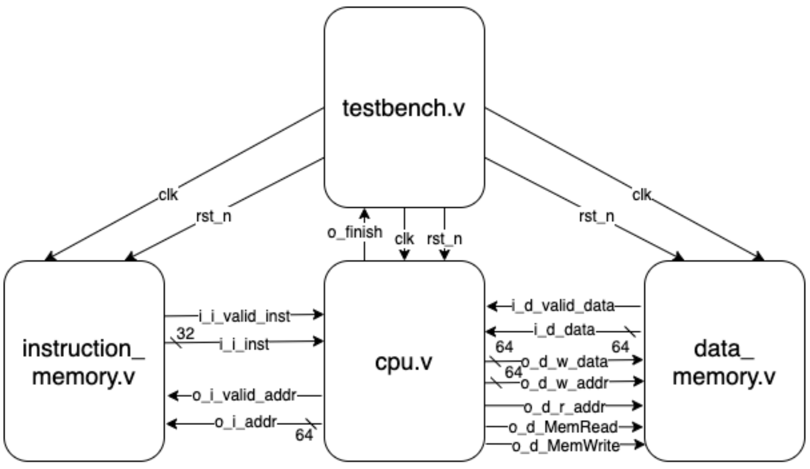
In this section, we are going to implement a pipeline cpu.
The provided instruction memory is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_valid	Input	1	Signal that tells pc-address from cpu is ready
i_addr	Input	64	64-bits address from cpu
o_valid	Output	1	Valid when instruction is ready
o_inst	Output	32	32-bits instruction to cpu

And the provided data memory is as follows:

Signal	I/O	Width	Functionality
i_clk	Input	1	Clock signal
i_rst_n	Input	1	Active low asynchronous reset
i_data	Input	64	64-bits data that will be stored
i_w_addr	Input	64	Write to target 64-bits address
i_r_addr	Input	64	Read from target 64-bits address
i_MemRead	Input	1	One cycle signal and set current mode to reading
i_MemWrite	Input	1	One cycle signal and set current mode to writing
o_valid	Output	1	One cycle signal telling data is ready (used when `ld` happens)
o_data	Output	64	64-bits data from data memory (used when `ld` happens)

The test environment is as follows:



The naming of the wire
is in the perspective of cpu

We will only test the instructions highlighted in the red box, as the figures below

imm[11:0]		rs1	110	rd	0000011	LWU
imm[11:0]		rs1	011	rd	0000011	LD
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	SD
000000	shamt	rs1	001	rd	0010011	SLLI
000000	shamt	rs1	101	rd	0010011	SRLI
010000	shamt	rs1	101	rd	0010011	SRAI
imm[11:0]		rs1	000	rd	0011011	ADDIW
0000000	shamt	rs1	001	rd	0011011	SLLIW
0000000	shamt	rs1	101	rd	0011011	SRLIW
0100000	shamt	rs1	101	rd	0011011	SRAIW
0000000	rs2	rs1	000	rd	0111011	ADDW
0100000	rs2	rs1	000	rd	0111011	SUBW
0000000	rs2	rs1	001	rd	0111011	SLLW
0000000	rs2	rs1	101	rd	0111011	SRLW
0100000	rs2	rs1	101	rd	0111011	SRAW

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

And one more instruction to be implemented is

i_inst	Function	Description
32'b11111111111111111111111111111111	`Stop`	Stop and set o_finish to 1

All the environment settings are the same as HW3 except the rule of accessing `data_memory.v` and `instruction_memory.v`, and the interface of modules are changed this time. See the supplementary.pdf for more information.

You may want to reference the diagram of pipelined cpu from textbook.

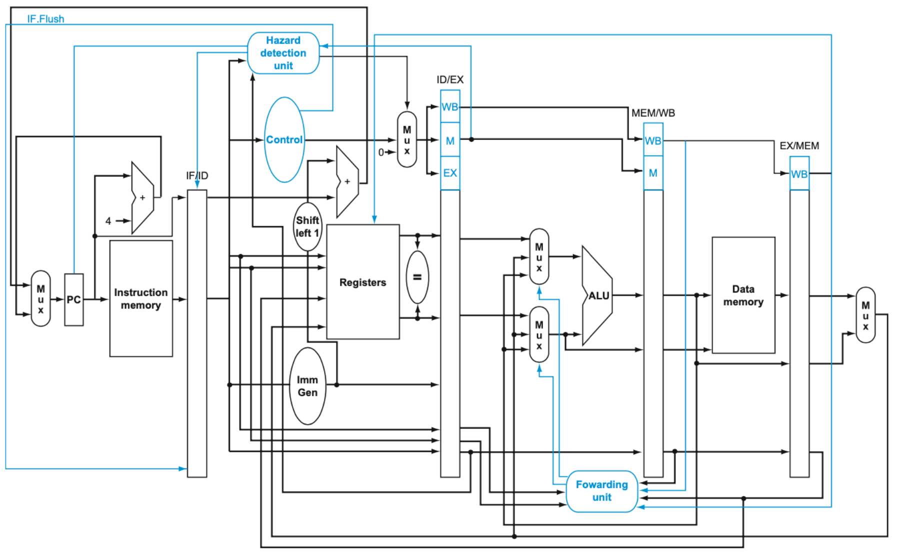


FIGURE 4.62 The final datapath and control for this chapter. Note that this is a stylized figure rather than a detailed datapath, so it's missing the ALUsrc Mux from Figure 4.55 and the multiplexor controls from Figure 4.49.

To make sure that pipeline is actually implemented in your design, we are going to use an open source synthesis tool Yosys to check the timing of the critical path in your design. We'll also use the FreePDK 45 nm process standard cell library provided here.

You can either build Yosys yourself or use the image provided

```
docker pull ntuca2022/hw4:version1 # size ~ 1.28G
docker run --name=test -it ntuca2022/hw4:version1
cd /root
ls
```

Folder structure for this homework:

```
HW4/
  |-- testcases/
  |     |-- generate.s
  |     `-- generate.cpp
  |-- codes/
  |     |-- cpu.v
  |     |-- data_memory.v         // provided data memory
  |     `-- instruction_memory.v  // provided instruction memory
  |-- testbench.v
  |-- Makefile
  |-- cpu.ys         // synthesis command
  `-- stdcells.lib   // FreePDK 45 nm standard cell library
```

Specify all the used modules in the **cpu.ys** file, then run

```
make       // Compile
make test  // Test all test cases
make time  // Show the timing and area used in your design
```

Information about your design is shown when running `make time`:

```
ABC: WireLoad = "none"  Gates =  13123 ( 14.8 %)   Cap =  3.2 ff (  1.9 %)
Area =    17519.56 ( 87.9 %)   Delay =  1091.13 ps  (  5.1 %)
```

You can optimize the cpu for the 3 workloads (code address range, data address range, etc), but it should not affect other test cases.

Grading:

- Correctness check (10%)
  - 10 testcases, each **2%** for correctness check
- Required area and frequency (inverse of delay) (32%)
  - Area < 25,000 μm^2, **and** frequency > 10MHz (5%)
  - Area < 25,000 μm^2, **and** frequency > 100MHz (5%)
  - Area < 25,000 μm^2, **and** frequency > 200MHz (5%)
  - Area < 25,000 μm^2, **and** frequency > 500MHz (5%)
  - Area < 25,000 μm^2, **and** frequency > 800MHz (4%)
  - Area < 25,000 μm^2, **and** frequency > 1000MHz (3%)
  - Area < 25,000 μm^2, **and** frequency > 1200MHz (3%)
  - Area < 25,000 μm^2, **and** frequency > 1500MHz (2%)
- Required time (clock cycle * operating frequency) to finish workloads from last 3 testcases. (32%)
  - Workload1 < 100,000 ns (5%)
  - Workload2 < 150,000 ns (5%)
  - Workload3 < 200,000 ns (5%)
  - Workload1 < 10,000 ns (5%)
  - Workload2 < 15,000 ns (4%)
  - Workload3 < 20,000 ns (3%)
  - Workload1 < 5,000 ns, **and** Workload2 < 20,000 ns, **and** Workload3 < 15,000 ns (3%)
  - Workload1 < 3,500 ns, **and** Workload2 < 9,000 ns, **and** Workload3 < 10,000 ns (2%)

Report (16%)

You can describe your pipeline design and how you did it and answer the following questions.

- What is the latency of each module in your design? (e.g. ALU, register_file) (2%)
- Which path is the critical path of your cpu? And how can you decrease the latency of it? (2%)
- How to solve data hazard? (3%)
- How to solve control hazard? (3%)
- Describe 3 different workloads attributes, and which one can be improved tremendously by branch predictor? (3%)
- Is it always beneficial to insert multiple stage of pipeline in designs? How does it affect the latency? (3%)

Submission

- Zip and upload your file to COOL in the following format:

```
Bxxxxxxxx/              <-- zip this folder
    |-- cpu.ys          // specify the used *.v file, not including testbench and memory
    |-- cpu.f           // specify the used *.v file, including testbench and memory
    |-- codes/          // put all your *.v file here, including cpu_syn.v
    `-- report.pdf      // report on your programming
```

- Late submission: (Total score)*0
- If there's any question, please send email to ntuca2022@gmail.com.
- TA hour for this homework: Thur 13:30 14:30 (We won't debug your codes!)

Deadline

- Deadline: 2022/12/25 23:59

Supplementary

- **HW4 introduction-video (2020)
- It's only for reference, since there are minor modifications on this homework.