Digital System Design

Final Project
Hardware Implementation of Pipelined RISC-V

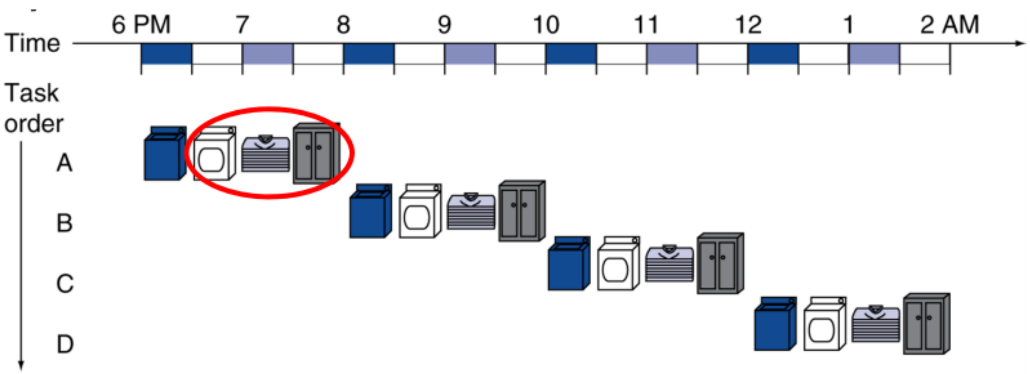
Speaker: Daniel

Instructor: 吳安宇教授

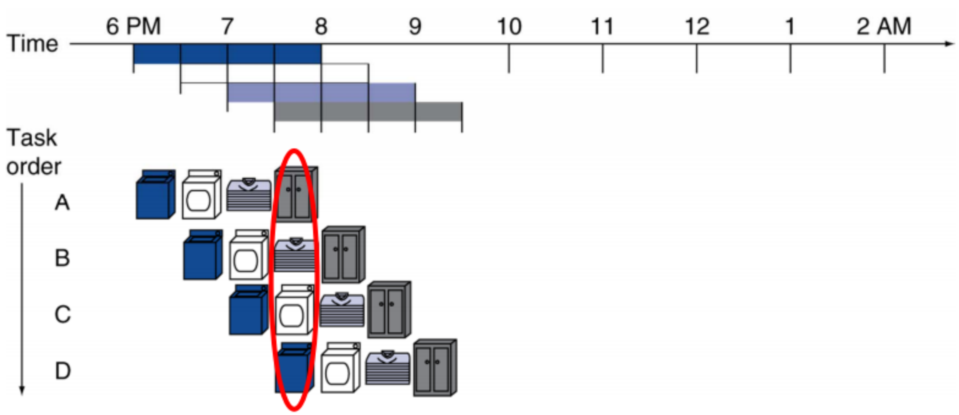
Date: 2023/05/16

RISC-V Processors

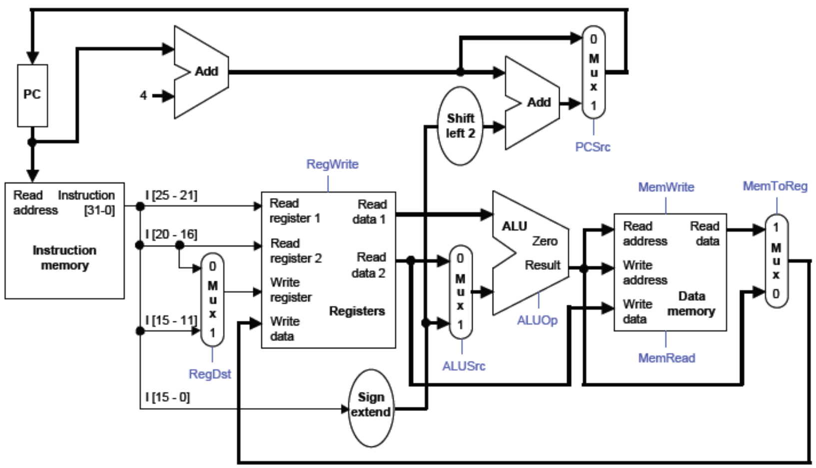
- Single-Cycle
  - Simple design with low throughput

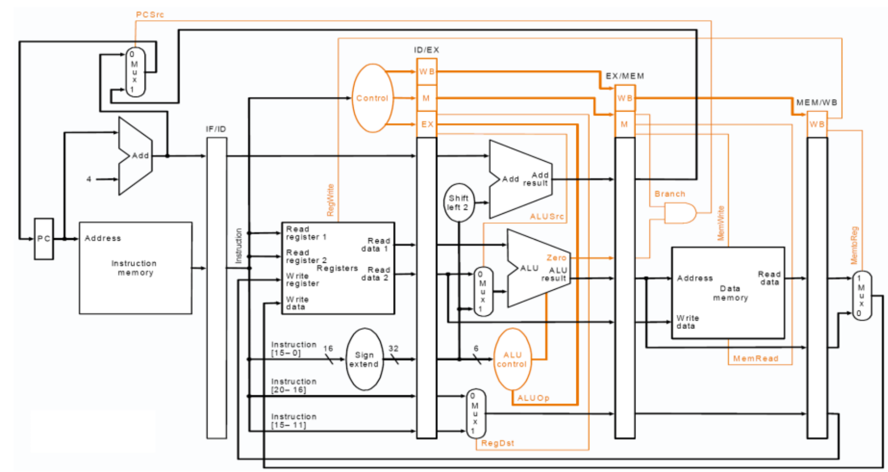


- Pipelined
  - Higher throughput
  - Complex design
    - For handle hazard



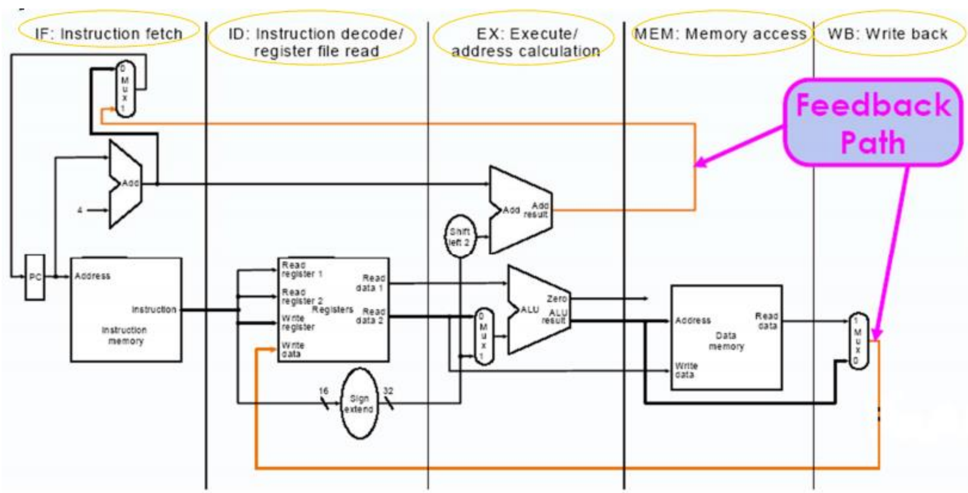
Final Project

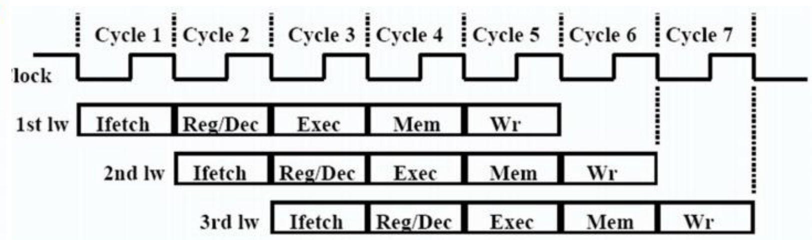




- ❖ HW2: Single Cycle RISC-V
- ❖ Final Project: Pipelined RISC-V Processor
  - ❖ With Instruction cache and data cache

Pipeline





- Splits into several functional unit and perform instruction in parallel way

- Your design should follow this 5-stage pipelined structure

Required Instruction Set

- You need to modify several parts to fit our specifications.
- For example, you need to add the path for **J-type instructions**

Table 1. Required Instruction Set

Name	Description
ADD	Addition, overflow detection for signed operand is not required*
ADDI	Addition immediate with sign-extension, without overflow detection*
SUB	Subtract, overflow detection for signed operand is not required*
AND	Boolean logic operation
ANDI	Boolean logic operation with 12bit of immediate
OR	Boolean logic operation
ORI	Boolean logic operation with 12bit of immediate
XOR	Boolean logic operation
XORI	Boolean logic operation with 12bit of immediate
SLLI	Shift left logical (zero padding)
SRAI	Shift right arithmetic (sign-digit padding)
SRLI	Shift right logical (zero padding)
SLT	Set less than, comparison instruction
SLTI	Set less than variable, comparison instruction
BEQ	Branch on equal, conditional branch instruction
BNE	Branch on not equal, conditional branch instruction
JAL	Unconditionally jump and link (Save next PC in $rd)
JALR	Jump and link register(Save next PC in $rd)
LW	Load word from data memory (assign word-aligned)
SW	Store word to data memory (assign word-aligned)
NOP	No operation(addi $r0 $r0 0)

* Different from definition in [1], the exception handler for arithmetic overflow is not required.

Hazards

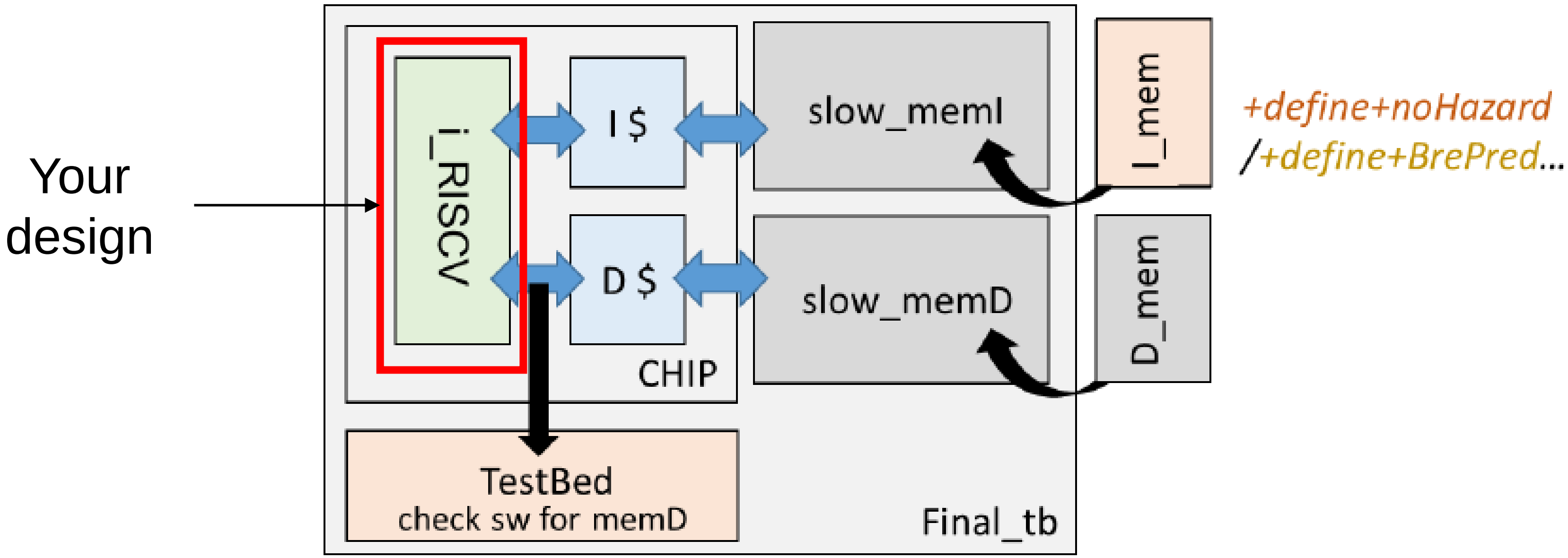
- In order to implement pipelined-RISC-V, you need to resolving hazard in hardware
  - Structural Hazard

	1	2	3	4	5
Load	Ifetch	Reg/Dec	Exec	Mem	Wr

	1	2	3	4
R-type	Ifetch	Reg/Dec	Exec	Wr

    - the hardware cannot support the combination of instructions
  - Data Hazard
    - data that is needed to execute the instruction is not yet available (dynamic).
  - Load-use Data Hazard
    - load instruction (lw) (read data from main memory) has not yet become available when it is requested
  - Control hazard (Branch hazard)
    - instruction that was fetched is NOT the one that is needed

How's the test works?



- I_mem: Instruction Memory
- D_mem: Data memory

EXTENSIONS

Extensions

- Branch Prediction(+define+BrPred)
  - Always not branch
  - Branch / Not branch / Branch
  - Always branch
- Supporting compressed instructions (+define+(de)compression)
  - Extract C-Instructions

Branch Prediction

- For loop is commonly see in programming
  - for(i=0;i<k;i++) → they did BEQ in every loop

- To increase overall efficiency of system, branch prediction is always used

```
ADD  $t0, $0, $0      #set $t0 to 0          for(int i=0; i != 100; i++)   {…}
ADDI$t1, $0, 100      #set $t1 to 100        for(int i=0; i != 100; i++)   {…}
SLL…                  #loop body             for(int i=0; i != 100; i++)   {…}
ADD…                  #assume this loop contains 3 instructions
SUB…
ADDI$t0, $t0, 1       #add 1 to $t0             for(int i=0; i != 100; i++)   {…}
BNE  $t0, $t1, -5     #branch if $t0 != $t1     for(int i=0; i != 100; i++)   {…}
```

With branch prediction → waste 3 cycles

Waste 99 cycles

Compressed Instruction

- Implement the following 16 C-instructions as the extension to your base RISC-V core

C.ADD	C.ANDI	C.LW	C.J
C.MV	C.SLLI	C.SW	C.JAL
C.ADDI	C.SRLI	C.BEQZ	C.JR
C.NOP	C.SRAI	C.BNEZ	C.JALR

- Extract information encoded in C-instructions
- PC increment
- Address alignment issued

Simulation for Baseline & Extensions

```
source /usr/cad/synopsys/CIC/vcs.cshrc
source /usr/spring_soft/CIC/verdi.cshrc
// RTL
vcs Final_tb.v CHIP.v slow_memory.v [other RTL files] -full64 -R -
debug_access+all +v2k +define+noHazard

// Gate level
vcs Final_tb.v CHIP_syn.v slow_memory.v -v tsmc13.v -full64 -R
-debug_access+all +v2k +define+noHazard +define+SDF
```

```
// For different condition (I_mem, TestBed)
`ifdef noHazard
    `define IMEM_INIT "I_mem_noHazard"
    `include "./TestBed_noHazard.v"
`endif
`ifdef hasHazard
    `define IMEM_INIT "I_mem_hasHazard"
    `include "./TestBed_hasHazard.v"
`endif
`ifdef BrPred
    `define IMEM_INIT "I_mem_BrPred"
    `include "./TestBed_BrPred.v"
`endif
`ifdef compression
    `define IMEM_INIT "I_mem_compression"
    `include "./TestBed_compression.v"
`endif
`ifdef decompression
    `define IMEM_INIT "I_mem_decompression"
    `include "./TestBed_compression.v"
`endif
```

- Final_tb.v in Baseline/src
- Change +noHazard for testing different cases
  - +hasHazard
  - +BrPred
  - +(de)compression

Simulation for Q_sort

```
source /usr/cad/synopsys/CIC/vcs.cshrc
source /usr/spring_soft/CIC/verdi.cshrc
// RTL
vcs Final_tb.v CHIP.v slow_memory.v [other RTL files] -full64 -R -
debug_access+all +v2k

// Gate level
vcs Final_tb.v CHIP_syn.v slow_memory.v -v tsmc13.v -full64 -R
-debug_access+all +v2k +define+SDF
```

```
`ifdef COMPRESS
    `define END_PC 320
    `define IMEM_INIT "I_mem_compression"
`else
    `define END_PC 400
    `define IMEM_INIT "I_mem"
`endif
```

- Final_tb.v in Q_sort
- If you want to run on compressed instructions
    - +define+COMPRESS

Grading Policy

- All grades of this project consist of two aspects:
  - 1) Baseline check point (40%)
    - Check Point Presentation (5%)
    - noHazard Gate Level (15%)
    - hasHazard Gate Level (20%)
  - 2) Final presentation and submission (60%)
    - Final Presentation (25%)
    - Branch Prediction Gate Level (5%)
    - Compressed Instr. Gate Level (5%)
    - Q_sort AT Ranking (15%)
    - Report (10%)

Baseline Check Point (40%)

- Checkpoint
  - 4-6 pages slides (about 5 minutes)
  - Confirm your current results and future plan
- Pass test programs (noHazard, hasHazard)
  - Supporting all instructions above
  - With caches
  - Complete the circuit synthesis. Note that the slack cannot be negative.

Final Presentation and Submission (60%)

- Each team should prepare a full talk (about 10-15 slides within 10 minutes) for your fantastic work on extension!

- Implement as much and deep as you can of the topics of extension
  - Deeper exploration ➔ higher score
  - The content also affect quality of presentation and report

- Then the AT performance is evaluated by Q_sort
  - Area (um2) * Total simulation time (ns)

※Notice

1. Latches are not allowed in gate level code after synthesis, use Flip-flop instead.
2. Negative Slack and Timing Violations are not allowed after synthesis.
3. There will be hidden cases for all test patterns.
4. The tsmc13.v file is not allowed to be downloaded! Or you may offend the copyright protected by NTU & TSRI!

Schedule

Date	Submission/Event
5/16	Final project announcement
5/30	A. Baseline checkpoint. Each team should prepare a presentation (*4-6 pages, about 5 minutes*) to confirm your current results and future plan. You should ***upload the results and slide to COOL before 1 pm***. B. Briefly show your design for passing Baseline testbenches (noHazard, hasHazard). C. Extension topics plan should be included in the presentation D. Please attach work assignment chart at last page
6/13	Final presentation. Each team should prepare a full talk (about 10-20 slides *within 15 minutes*) to demonstrate your fantastic work! Detail presentation plan will be announced on COOL.
6/17	Final submission, including a detailed report (*8-16 pages*), the presentation slides, and all the source codes (including all the RTL code and synthesis related files: *.v, *.sdf, *.ddc and a Readme.txt). You should upload the final submission to COOL by the day.