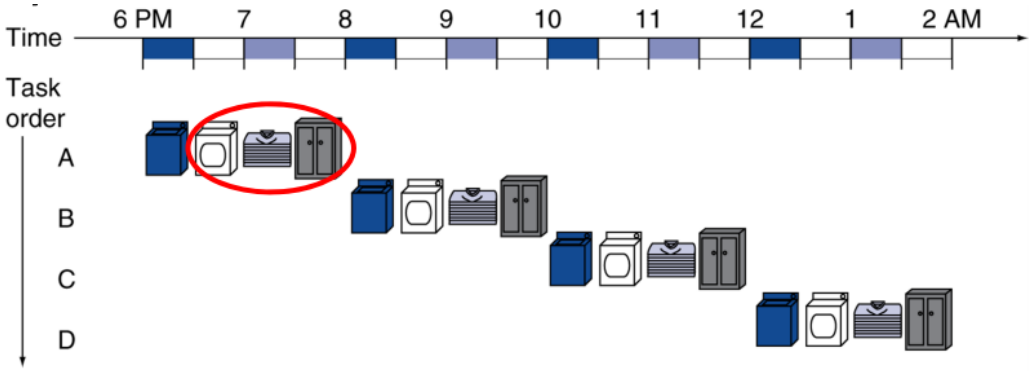
Digital System Design

Final Project
Hardware Implementation of Pipelined MIPS and RISC-V

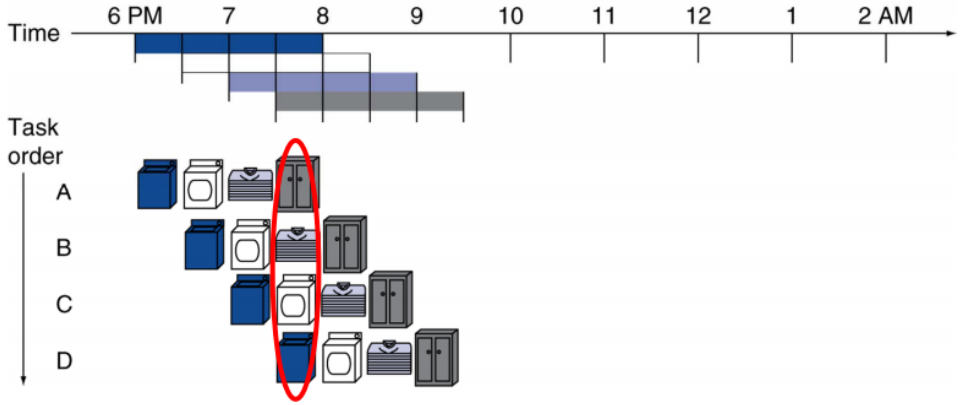
Speaker: Kane
Instructor: 吳安宇教授
Date: 2020/05/28

MIPS/RISC-V Processors

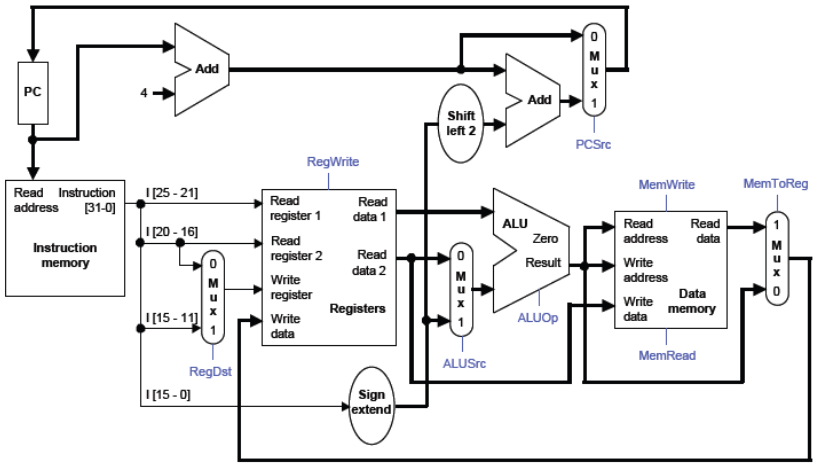
- Single-Cycle
  - Simple design with low throughput

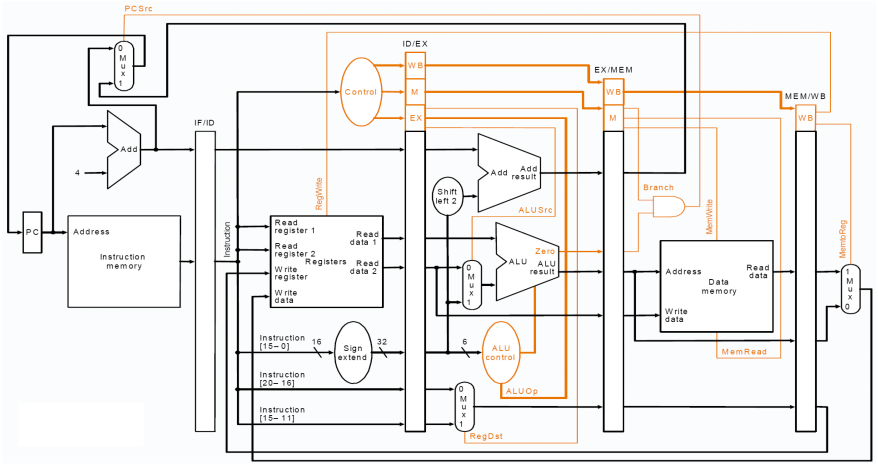


- Pipelined
  - Higher throughput
  - Complex design
    - For handle hazard

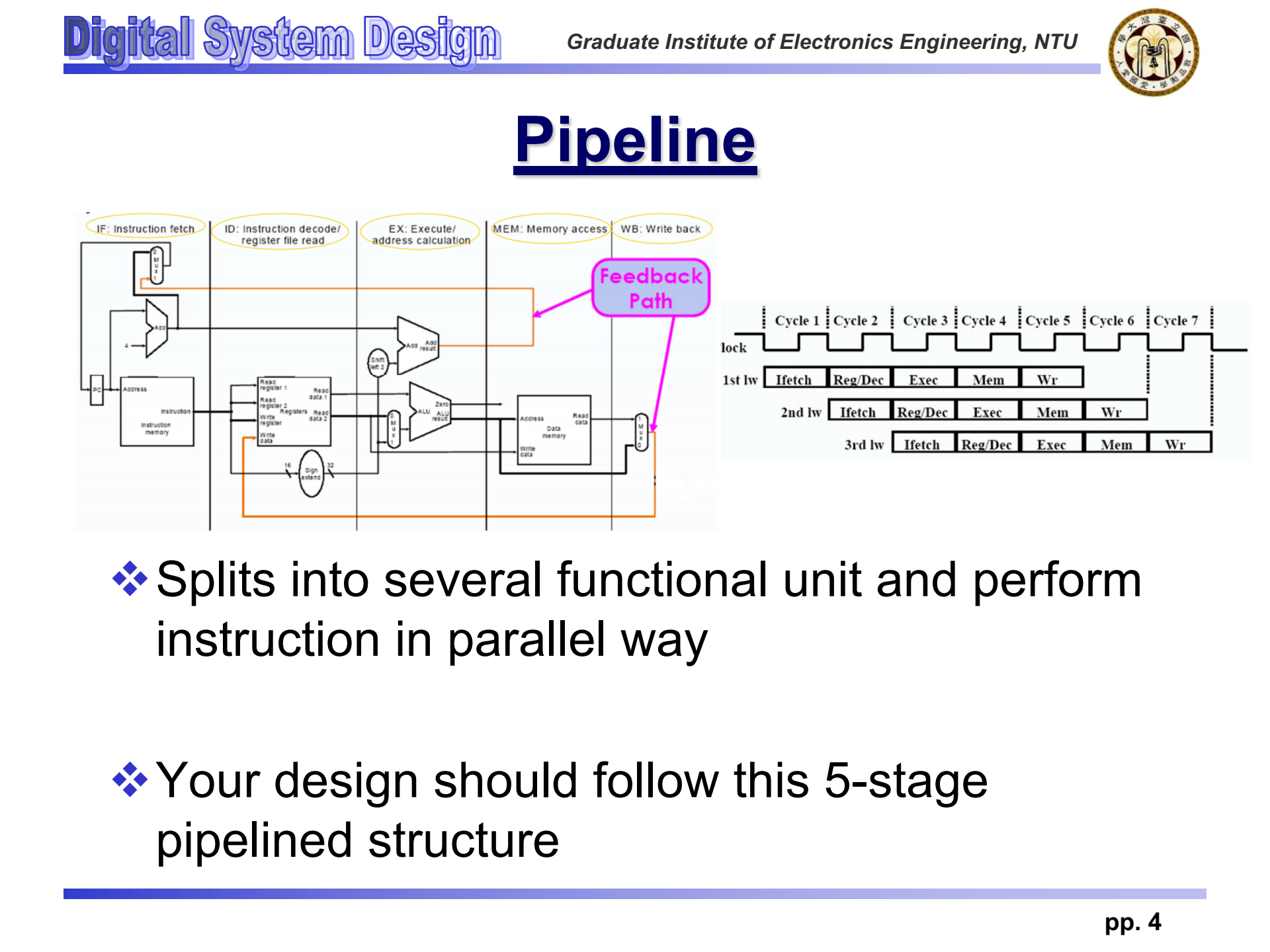


Final Project





- HW3: Single Cycle MIPS/RISC-V
- Final Project: Pipelined MIPS/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 pars to fit our specifications.
- For example, you need to add the path for **J-type instructions**

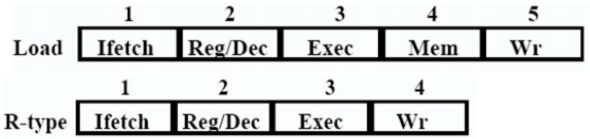
Table 1. Required Instruction Set

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, zero-extension for upper 16bit of immediate
OR	Boolean logic operation
ORI	Boolean logic operation, zero-extension for upper 16bit of immediate
XOR	Boolean logic operation
XORI	Boolean logic operation, zero-extension for upper 16bit of immediate
NOR	Boolean logic operation
SLL	Shift left logical (zero padding)
SRA	Shift right arithmetic (sign-digit padding)
SRL	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
J	Unconditionally jump
JAL	Unconditionally jump and link (Save next PC in $r31)
JR	Unconditionally jump to the instruction whose address is in $rs
JALR	Jump and link register
LW	Load word from data memory (assign word-aligned)
SW	Store word to data memory (assign word-aligned)
NOP	No operation

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

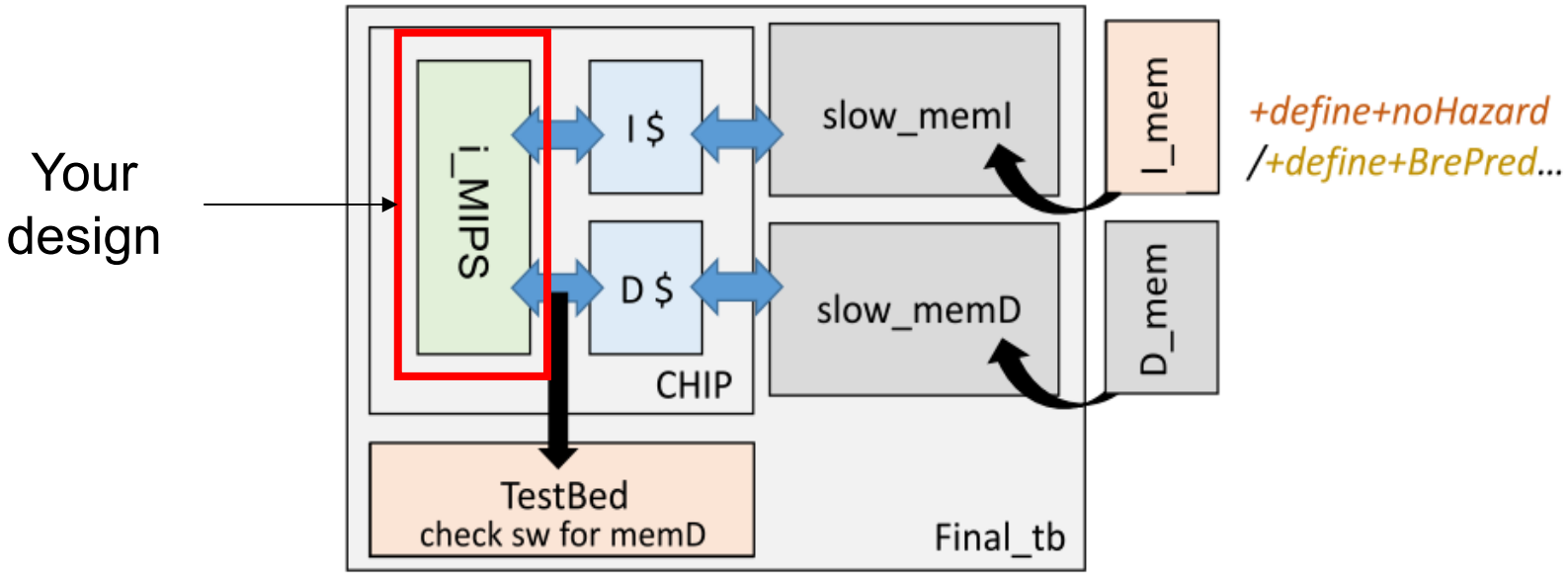
Hazards

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

How's the test works?



- I_mem: Instruction Memory
- D_mem: Data memory

Extensions (MIPS)

- Branch Prediction(+define+BrPred)
  - Always not branch
  - Branch / Not branch / Branch
  - Always branch
- L2 Cache(+define+L2Cache)
  - Long version of Fibonacci
- Multiplication/Division (+define+MultDiv)
  - Factorial

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++)   {…}
```

Waste 99 cycles

With branch prediction → waste 3 cycles

Comparison Metrics

- *Score 1 (BP_S1): Total execution cycles of 'I_mem_BrPred'*
  - BP_S1 = total cycle counts of the I_mem_BrPred
- *Score 2 (BP_S2): Total execution cycles of 'I_mem_hasHazard'*
  - BP_S2 = total cycle counts of the I_mem
- *Score 3 (BP_S3): Synthesis area of BPU (um2)*

L2 Cache

- Main goal
  - Add 2 separate L2 caches for I-cache and D-cache
  - Unified L2 cache (1 L2 cache for both L1 I-cache & L1 D-cache) will be considered as better design (not required)
- Total size of respective L1 cache is suggested to be 32 words
  - 8 blocks & each 4 words
- Total size of respective L2 caches(I+D) are suggested to be 256 words
  - Block size and number of blocks are up to your considerations

Comparison Metrics

- Base on the test program: "I_mem_L2Cache"
- Score 1 (L2C_S1): Avg. memory access time (ns)
  - L2C_S1 = HT1 + MR1*(HT2+MR2*MP2)
  - HT: hit time
  - MR: miss rate
  - MP: miss penalty
  - ~1: of level 1 cache
  - ~2: of level 2 cache
- Score 2 (L2C_S2): Total execution time (ns)
  - L2C_S2 = total execution time of the test program

- Don't worry about the performance evaluation. It is just one of the criteria. **Focus more on what you design to solve problems you face.**

Multiplication/Division

- Main goal
  - Add a computation unit and control unit to support multiplication and division instructions

Instrcution	op/func	Meaning
mult $rs $rt	0/24	Multiply $rs with $rt, and store upper 32 bits in $HI, lower 32 bits in $LO
div $rs $rt	0/26	Divide $rs by $rt, and store the remainder in $HI, the quotient in $LO
mfhi $rd	0/16	Move the data from $HI to $rd
mflo $rd	0/18	Move the data from $LO to $rd

Comparison Metrics

- Base on the test program "I_mem_MultDiv"
- Score 1 (MD_S1): Area of MultDiv (um^2)
  - MD_S1 = area of MultDiv - area of baseline chip
- Score 2 (MD_S2): Total execution time (ns)
  - MD_S2 = total execution time of test program
- Score 3 (MD_S3): Minimum clock period (ns)
  - MD_S3 = clock period of MIPS core

- Don't worry about the performance evaluation. It is just one of the criteria. **Focus more on what you design to solve problems you face.**

Extensions (RISC-V)

- Branch Prediction(+define+BrPred)
  - Always not branch
  - Branch / Not branch / Branch
  - Always branch
- L2 Cache(+define+L2Cache)
  - Long version of Fibonacci
- Supporting compressed instructions (+define+(de)compression)
  - Extract C-Instructions
  - PC increment
  - Address alignment issues

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

Comparison Metrics

- A(compressed design – baseline design) *T(given compressed testbench)
- Some comparison of different method can be discuss in presentation or in report

Simulation Example

```
source /usr/cad/cadence/cshrc
source /usr/spring_soft/CIC/verdi.cshrc
```

```
ncverilog Final_tb.v CHIP.v slow_memory.v +define+noHazard +access+r
```

```
ncverilog Final_tb.v CHIP_syn.v slow_memory.v tsmc13.v
+define+noHazard +define+SDF +access+r
```

```
// 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 L2Cache
    `define IMEM_INIT "I_mem_L2Cache"
    `include "./TestBed_L2Cache.v"
`endif
`ifdef MultDiv
    `define IMEM_INIT "I_mem_MultDiv"
    `include "./TestBed_MultDiv.v"
`endif
```

- Change +noHazard for testing different cases
  - +hasHazard
  - +BrPred
  - +L2Cache
  - +MultDiv

Grading Policy

- All grades of this project consist of three equal aspects:
  - 1) Proposal: Project check point (5%)
  - 2) Presentation (45%)
  - 3) Final Report: technical features (50%)

- The technical features are grading by two parts:
  - Baseline (50%)
  - Extension (50%)

Baseline (50%)

- The solid requirements include:
  - Supporting all instructions above
  - With caches
  - Pass all test assembly programs
  - Complete the circuit synthesis. Note that the slack cannot be negative.
- Then the performance is evaluated by:
  - **Area (um^2) * Total simulation time (ns);**
- And baseline points will be based on your AT ranking among other MIPS teams.

Extension(50%)

- There are three topics of extension.
  - Branch prediction mechanism.
  - Two-level caches, i.e. with L2 caches.
  - Supporting multiplication and division.
- Implement **as much and deep as you can** of the topics of extension.
- Deeper exploration → higher score
  - The content also affect quality of presentation

Presentation Hint

- More performance from deeper pipelines, parallelism (Chap. 4.10, page 332) may explore in your final project reports!
  - Branch prediction
  - Multiple issues
  - VLIW (Very long length Instruction Word)
  - Superscalar
  - Dynamic scheduling
  - Out-of-order execution
  - Speculation
  - Reorder buffer
  - Register renaming

※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. The tsmc13.v file is not allowed to be downloaded! Or you may offend the copyright protected by NTU & CIC!

Find your partner and topic

- Choose the topic you want to do, due 2020/05/30 24:00 (only team leader)
  - https://forms.gle/DP4hVreFQEr5oGJr9

Timeline

Date	Submission/Event
5/28	Final project announcement
5/30	Find your teammates and choose MIPS or RISCV (Fill out the Google form)
6/11	A. Checkpoint. Each team should prepare a proposal (*4-6 pages powerpoint (about 5 minutes)*) to confirm your current results and future plan. You should upload the team proposal to the CEIBA by the day. B. Extension topics plan should be included in the proposal C. Please attach work assignment chart at last page
6/29	Final presentation. Each team should prepare a full talk (*within 15 minutes*, about 10-20 slides) to demonstrate your fantastic work! Detail presentation plan will be announced in the CEIBA.
6/30	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 the CEIBA by the day.