



Computer Architecture

Final Project: Single Cycle CPU

TA: 黃士豪 (Shih-hao Huang)

Due: 2022/12/26 (Mon.) 23:59 (UTC+8)

Email: r10943004@ntu.edu.tw

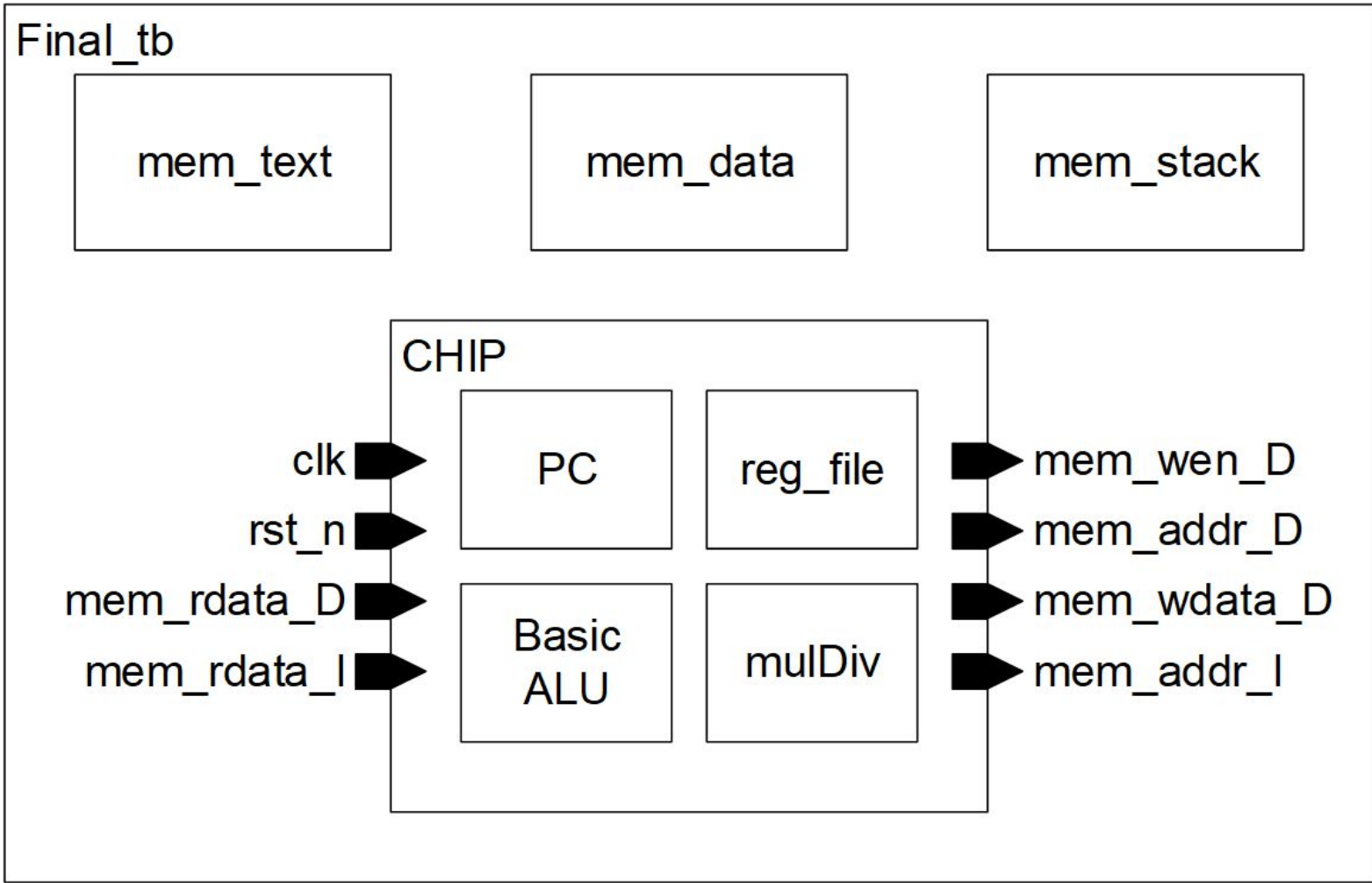


Goal

- ◆ Implement a single cycle CPU
- ◆ Add multiplication/division unit (mulDiv) to CPU (HW2)
- ◆ Handle multi-cycle operations
- ◆ Get more familiar with assembly and Verilog



Specification





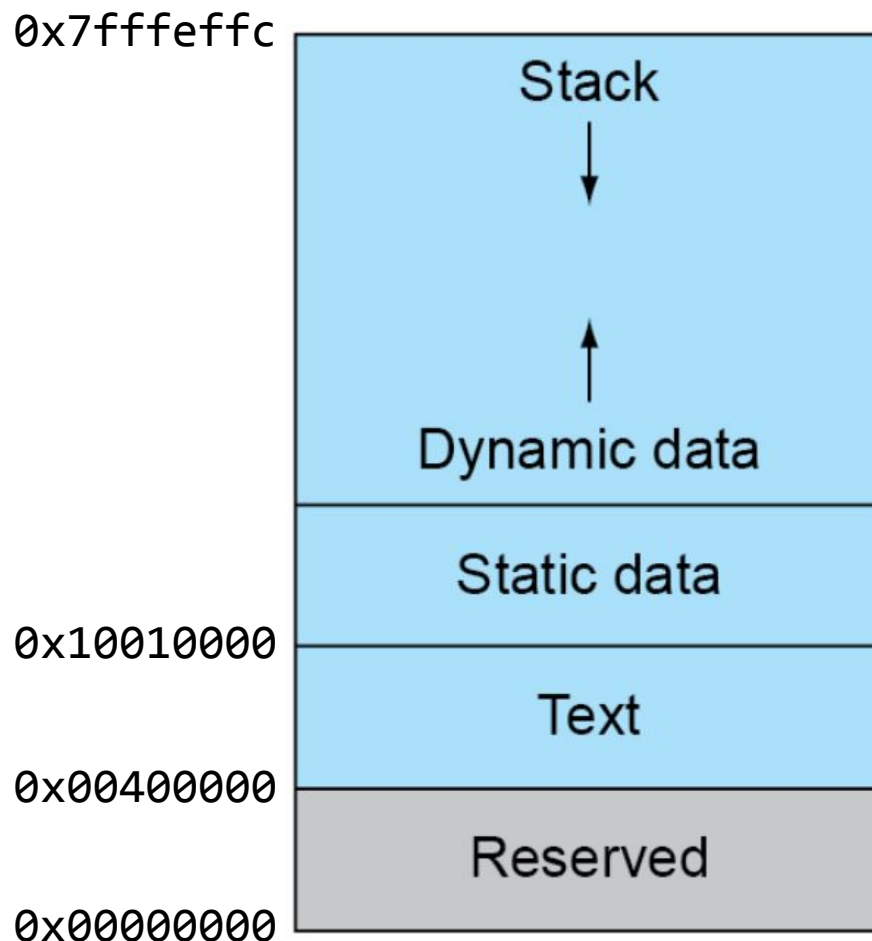
Port Definition

Name	I/O	Width	Description
clk	I	1	Positive edge-triggered clock
rst_n	I	1	Asynchronous negative edge reset
mem_wen_D	O	1	0: Read data from data/stack memory 1: Write data to data/stack memory
mem_addr_D	O	32	Address of data/stack memory
mem_wdata_D	O	32	Data written to data/stack memory
mem_rdata_D	I	32	Data read from data/stack memory
mem_addr_I	O	32	Address of instruction (text) memory
mem_rdata_I	I	32	Instruction read from instruction (text) memory



Memory Layout (1/2)

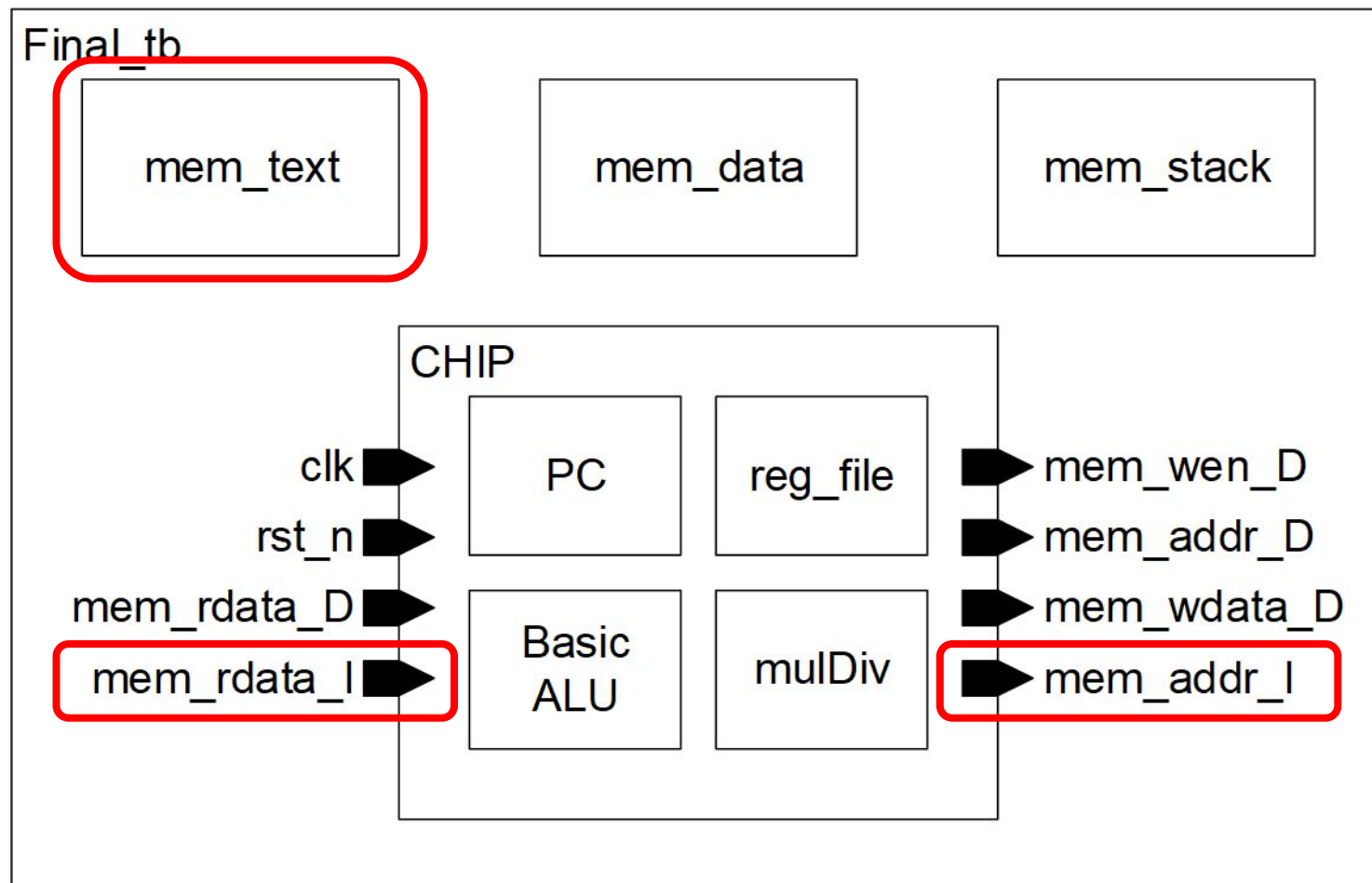
- ◆ In RARS simulator
- ◆ Text
 - ◆ Program code
- ◆ Data
 - ◆ Variables, arrays, etc.
- ◆ Stack
 - ◆ Automatic storage





Relate Memory to Testbench (1/4)

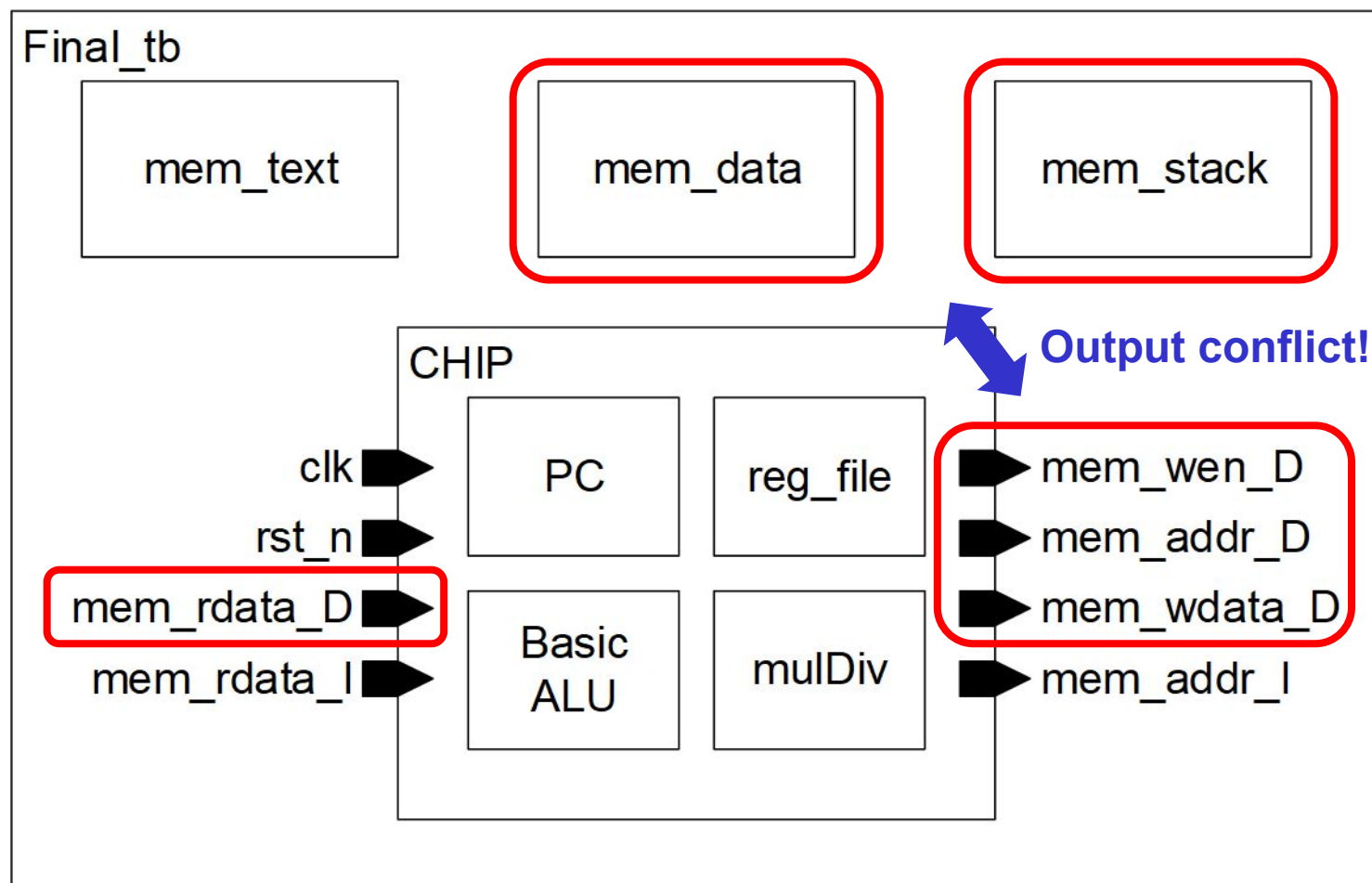
- ◆ Instruction (text) memory





Relate Memory to Testbench (2/4)

- ◆ Data/stack memory





Relate Memory to Testbench (3/4)

- ◆ Reduce size of memory blocks to improve simulation speed



```
`define SIZE_TEXT 36
`define SIZE_DATA 36
`define SIZE_STACK 36
```

- ◆ Define offset address for each memory block



- ◆ Define high impedance to avoid output conflict



- ◆ **Not synthesizable coding style!**

```
module memory #(
    parameter BITS = 32,
    parameter word_depth = 32
) (
    clk,
    rst_n,
    wen,
    a,
    d,
    q,
    offset
);
```

```
always @(*) begin
    q = {(BITS){1'bz}};
    for (i=0; i<word_depth; i=i+1) begin
        if (mem_addr[i] == a)
            q = mem[i];
    end
    if (wen) q = d;
end
```

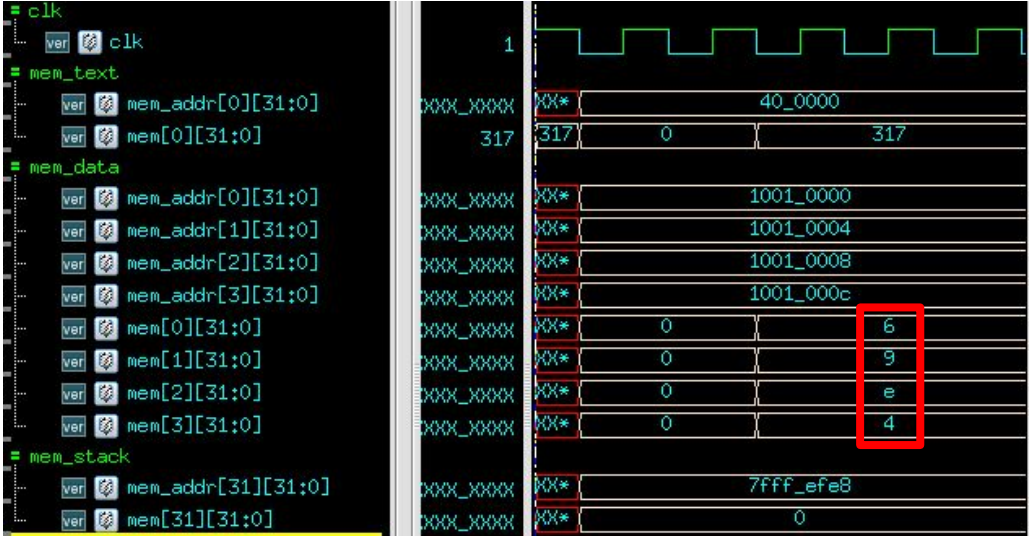



Relate Memory to Testbench (4/4)

- ◆ In RARS
- ◆ In Testbench

Text Segment			
Bkpt	Address	Code	Ba
<input type="checkbox"/>	0x00400000	0x00000317 auipc x6,0	
<input type="checkbox"/>	0x00400004	0x02430067 jalr x0,x6,36	
<input type="checkbox"/>	0x00400008	0x00b542b3 xor x5,x10,x11	
<input type="checkbox"/>	0x0040000c	0x00c5c333 xor x6,x11,x12	

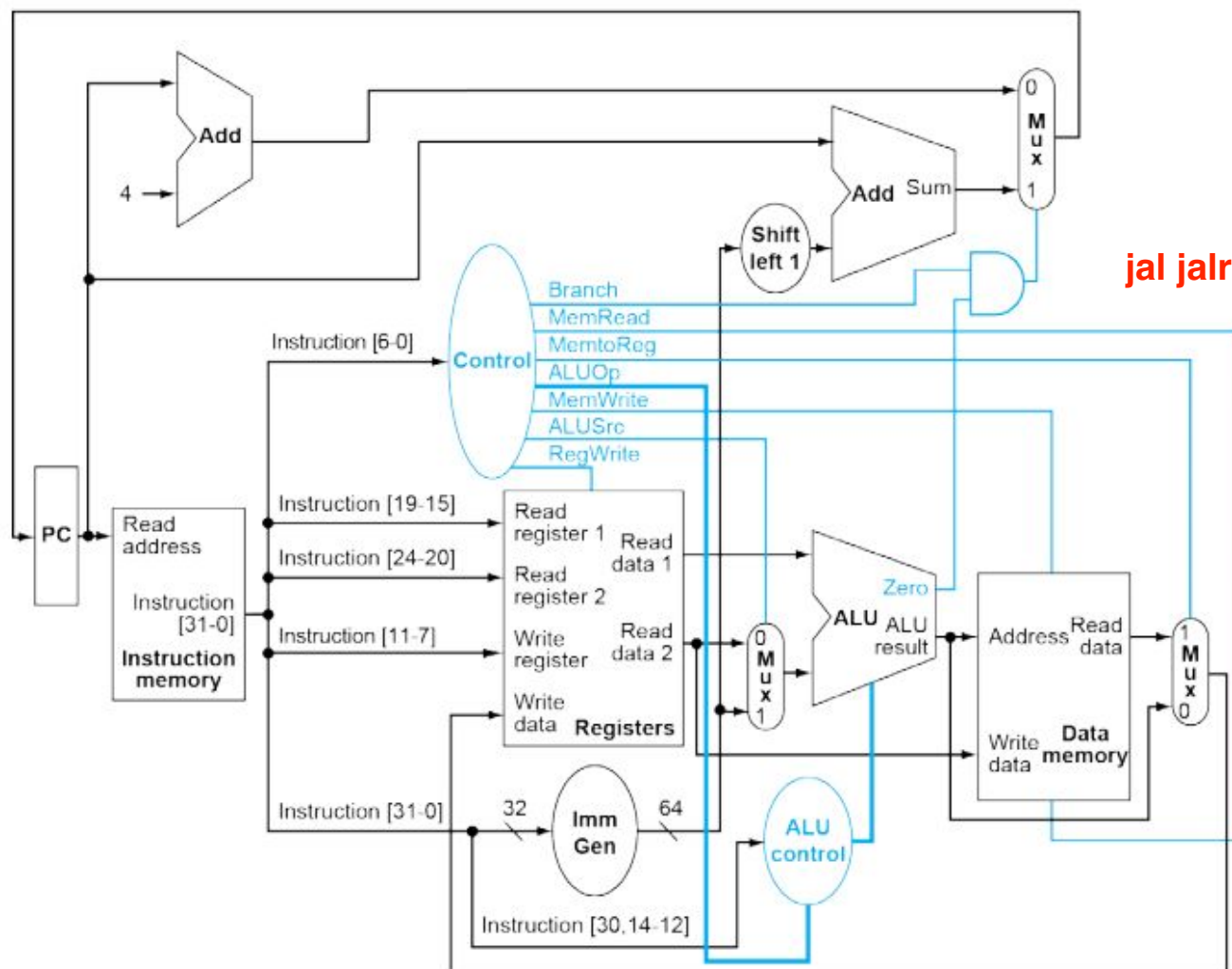
Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000006	0x00000009	0x0000000e	0x00000004
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000





Architecture

- ◆ Not complete (does not include jal, jalr, ...)



jal jalr要自己去slide看

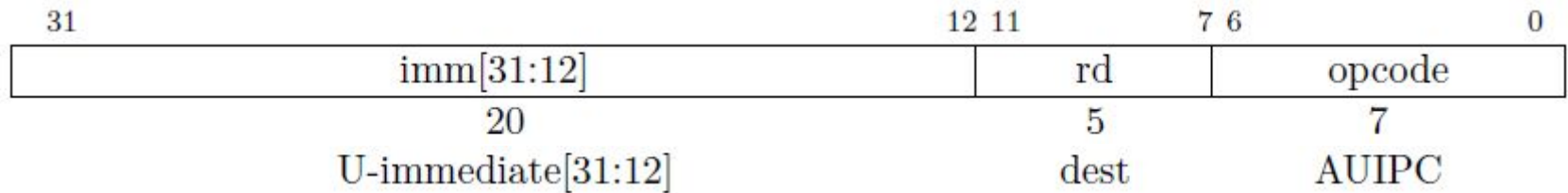


Supporting Instructions

- ◆ Your design must **at least** support
 - ◆ `auipc, jal, jalr`
 - ◆ `beq, lw, sw`
 - ◆ `addi, slti, add, sub, xor`
 - ◆ `mul`
- ◆ For **bonus** challengers
 - ◆ `bge, srai, slli ...` (you might have to use these instructions to finish your bonus)
- ◆ See “`Instruction_Set_Listings.pdf`” for more information of machine code



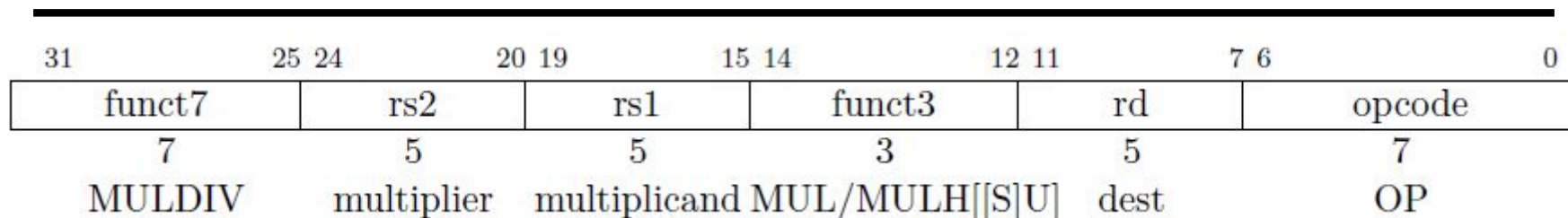
Supplement: Instruction “auipc”



- ◆ Add upper immediate to PC, and store the result to rd
 - ◆ auipc rd, U-immediate $pc + imm * (2^4)$
通常用於jump的前置動作
- ◆ Example: auipc x5, 1 (PC = 0x0001001c)
 - ◆ $0x0001001c + 0x00001000 = 0x0001101c$
 - ◆ Store 0x0001101c in x5



Supplement: Instruction “mul”

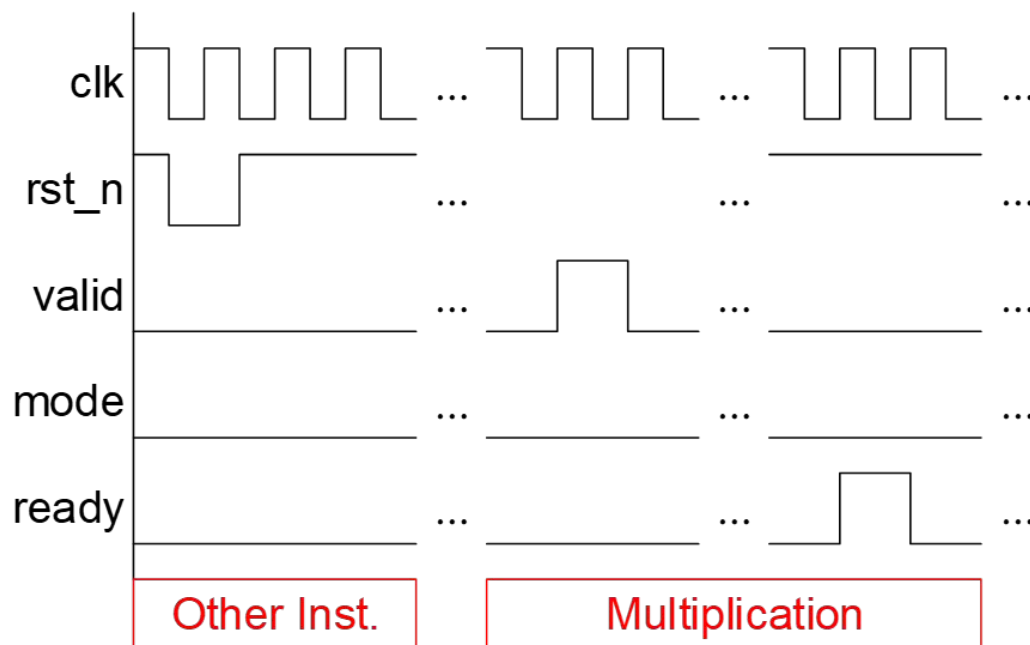


- ◆ Not included in RV32I
- ◆ Store the lower 32-b result ($rs1 \times rs2$) to rd
- ◆ Example: mul x10, x10, x6
 - ◆ $x10 = 0x00000001$, $x6 = 0x00000002$
 - ◆ $0x00000001 \times 0x00000002 = 0x00000002$
 - ◆ Store $0x00000002$ in x10
- ◆ **Your mulDiv can support this instruction!**



Multi-Cycle Operation

- ◆ Once CPU decodes mul operation, issue valid to your mulDiv
- ◆ Once CPU receives ready, store the lower 32-b result to rd
- ◆ You might have to design FSM in your CPU





Test Pattern 1: Leaf

- ◆ Modified from lecture slides
- ◆ The procedure loads a,b,c,d from 0x10010000–0x1001000c, and stores the result to 0x10010010
- ◆ Run simulation:
 - ◆ \$ ncverilog Final_tb.v +define+leaf +access+r

```
def leaf(a,b,c,d):
    f = (a^b) + (b^c) - (c^d)
    return f
```

xor

```
.data
a: .word 6
b: .word 9
c: .word 14
d: .word 4
```

Data Segment					
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	6	9	14	4	12
0x10010020	0	0	0	0	0
0x10010040	0	0	0	0	0
0x10010060	0	0	0	0	0
0x10010080	0	0	0	0	0
0x100100a0	0	0	0	0	0
0x100100c0	0	0	0	0	0
0x100100e0	0	0	0	0	0
0x10010100	0	0	0	0	0



Test Pattern 2: Perm

- ◆ Modified from lecture slides
- ◆ The procedure loads n,r from **遞迴function** 0x10010000–0x10010004, and stores the result to 0x00010008
- ◆ Run simulation:
 - ◆ \$ ncverilog Final_tb.v +define+perm +access+r

```
def perm(n,r):
    if r < 1:
        return 1
    else:
        return n*perm(n-1,r-1)
```

```
.data
    n: .word 10
    r: .word 3
```

Data Segment			
Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	10	3	720
0x10010020	0	0	0
0x10010040	0	0	0
0x10010060	0	0	0
0x10010080	0	0	0
0x100100a0	0	0	0
0x100100c0	0	0	0
0x100100e0	0	0	0
0x10010100	0	0	0



(Bonus) Test Pattern 3: (1/4)

◆ Design your assembly first

更複雜的遞迴
assembly要自己寫

- ◆
$$T(n) = \begin{cases} 2 \times T\left(\left\lfloor \frac{3n}{4} \right\rfloor\right) + \lfloor 0.875n \rfloor - 137, & n \geq 10 \\ 2 \times T(n-1), & 1 \leq n < 10 \\ 7, & n = 0 \end{cases}$$
- ◆ Example: $T(11) = 3456$, $T(30) = 55489$
- ◆ **Use recursive function**

```
FUNCTION:
    # Todo: Define your own function

# Do NOT modify this part!!!
__start:
    la    t0, n
    lw    x10, 0(t0)
    jal   x1, FUNCTION
    la    t0, n
    sw    x10, 4(t0)
    addi  a0, x0, 10
    ecall
```



(Bonus) Test Pattern 3: (2/4)

◆ Dump text memory file (Hexadecimal format)

step 2

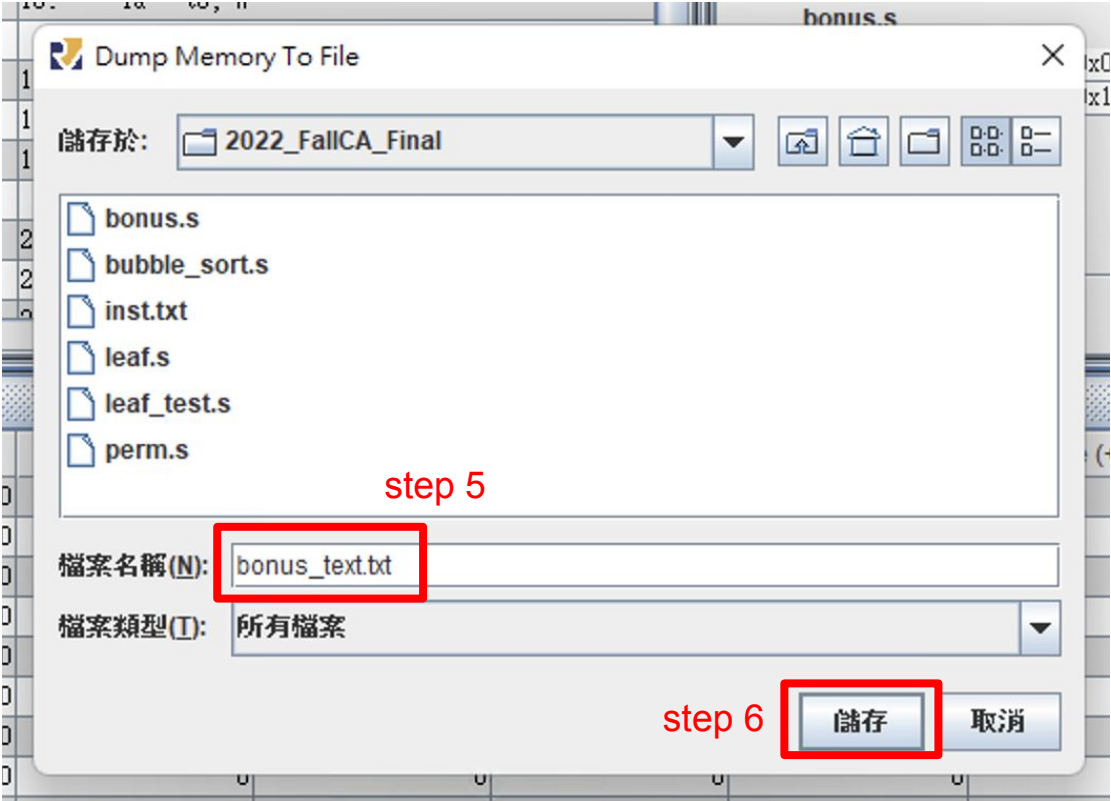
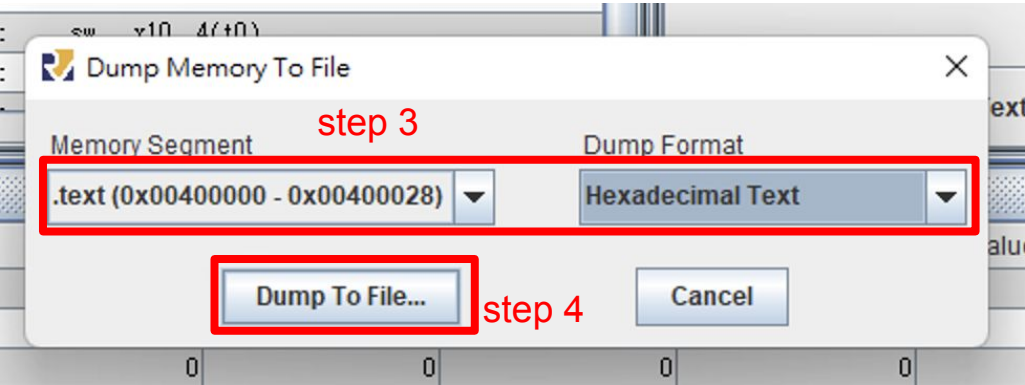
The screenshot shows a debugger interface with the following components:

- File Menu:** Open, Save, Save as, Dump Memory (highlighted with a red box and labeled 'step 2'), Exit.
- Code Window:** Displays assembly code with columns for Code, Basic, and Source. The first line is highlighted in yellow.
- Labels Window:** Shows labels and addresses, including (global), _start, bonus.s, FUNCTION, and n.
- Data Segment Window:** Displays memory addresses and their values in hexadecimal. The 'Hexadecimal Values' checkbox is checked (highlighted with a red box and labeled 'step 1').

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000b	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



(Bonus) Test Pattern 3: (3/4)





(Bonus) Test Pattern 3: (4/4)

- ◆ Save the binary file as: `./Verilog/bonus/bonus_text.txt`
 - ◆ Modify the text file: delete the last 2 instructions
- ◆ Test pattern generation: `./Verilog/bonus/bonus_gen.py`
- ◆ Run simulation:
 - ◆ `$ ncverilog Final_tb.v +define+bonus +access+r`

```
00000317|
00830067
0fc10297
ff828293
0002a503
ff5ff0ef
0fc10297
fe828293
00a2a223
```

Delete

```
00a00893
00000073
```

system call



Pattern Generation

- ◆ Three python codes provided:
 - ◆ leaf_gen.py
 - ◆ perm_gen.py
 - ◆ bonus_gen.py

- ◆ TA will change the variables in *_gen.py to generate new test patterns when testing your CPU design



Coding Style Check

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
alu_in_reg	Flip-flop	32	Y	N	Y	N	N	N	N
counter_reg	Flip-flop	5	Y	N	Y	N	N	N	N
shreg_reg	Flip-flop	64	Y	N	Y	N	N	N	N
state_reg	Flip-flop	2	Y	N	Y	N	N	N	N

- ◆ All sequential elements must be **flip-flops**
- ◆ Check by Design Compiler
- ◆ Command:
 - ◆ `$ dv -no_gui`
 - ◆ `design_vision> read_verilog CHIP.v`
- ◆ Exit:
 - ◆ `design_vision> exit`



Report

- ◆ Briefly describe your CPU architecture
- ◆ Describe how you design the data path of instructions not referred in the lecture slides (jal, jalr, auipc, ...)
- ◆ Describe how you handle multi-cycle instructions (mul)
- ◆ Record total simulation time (CYCLE = 10 ns)
 - ◆ Leaf: $a = 3, b = 9, c = 5, d = 17$
 - ◆ Perm: $n = 8, r = 5$
 - ◆ (Bonus: $n = 11$)

```
Simulation complete via $finish(1) at time 4795 NS + 0
```

- ◆ Describe your observation
- ◆ Snapshot the “Register table” in Design Compiler (p. 22)
- ◆ List a work distribution table



Submission

- ◆ Deadline: 12/26 (Mon.) 23:59
 - ◆ Late submission: 20 % reduction per day
- ◆ Upload Final_group_<group_id>.zip to ceiba
 - ◆ Final_group_<group_id>.zip
 - Final_group_<group_id>/
 - Final_group_<group_id>/CHIP.v
 - (Final_group_<group_id>/bonus.s)
 - (Final_group_<group_id>/bonus_text.txt)
 - Final_group_<group_id>/report.pdf
 - ◆ Wrong format: 20% reduction
- ◆ Example

```
[r10004@sullivan 2022fall_CA_Final_Grading]$ unzip Final_group_0.zip
Archive:  Final_group_0.zip
  creating: Final_group_0/
  inflating: Final_group_0/bonus.s
  inflating: Final_group_0/bonus_text.txt
  inflating: Final_group_0/CHIP.v
  extracting: Final_group_0/report.pdf
```




Score

- ◆ Simulation: 70 % (+ bonus 20 %)
 - ◆ Leaf
 - Default: 15 %
 - Change test pattern: 15 %
 - ◆ Perm
 - Default: 20 %
 - Change test pattern: 20 %
 - ◆ Bonus
 - Default: 10 %
 - Change test pattern: 10 %
- ◆ Report: 30 %
 - ◆ Content: 20 %
 - ◆ Snapshots: 5 %
 - ◆ Work distribution: 5 %