Lab4
Single cycle CPU

TA-黃炫峰

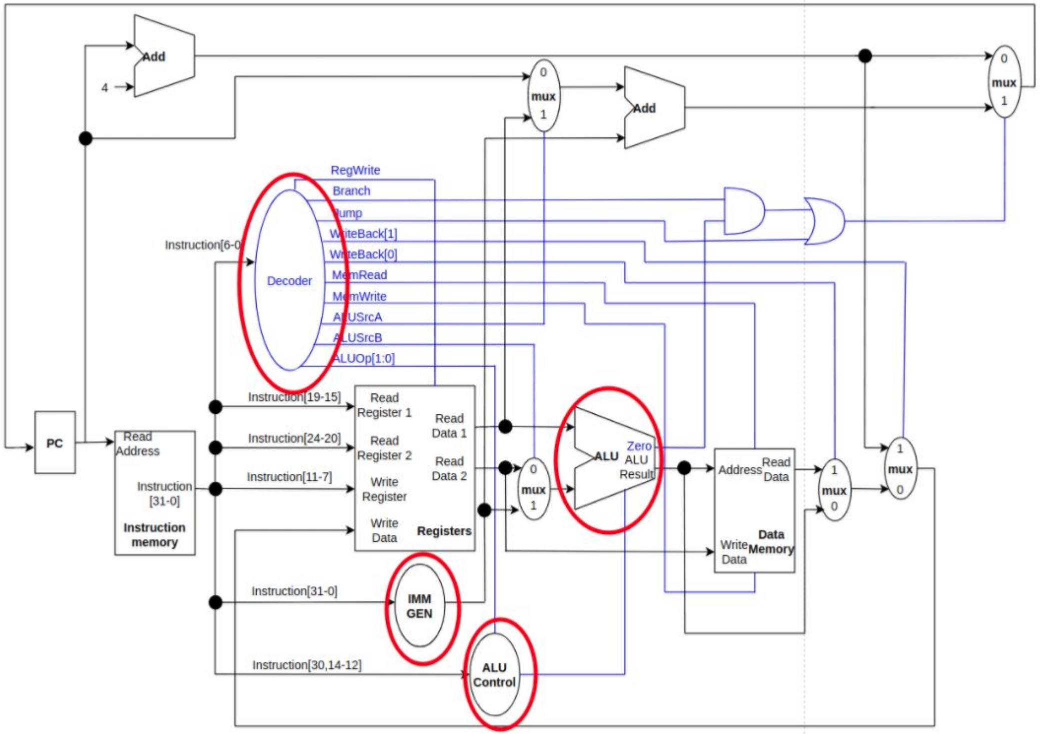
ma880521＠gmail.com

Object

- To realize how to set the control signal in different instruction type.(Decoder & ALU Controller)
- To clarify how sign-extend work.
- Connect all datapath to form a single cycle CPU.

Implement

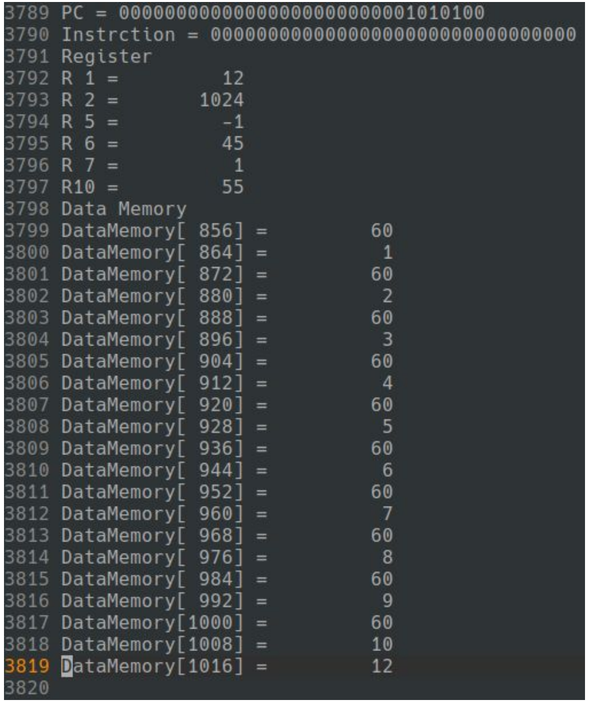
- You have to implement following unit:
  - Decoder
  - Sign-extend Unit
  - ALU Controller
  - ALU
  - Single Cycle CPU(Connect all unit)



Attached file

- **TODO**
  - **Lab4_release**
    - Decoder.v
    - Imm_Gen.v
    - ALU_Ctrl.v
    - alu.v
    - Simple_Single_CPU.v
- **Validate the correction of your implementation**
  - $ ./demo.sh
- **Debug**
  - result.txt
  - answer.txt

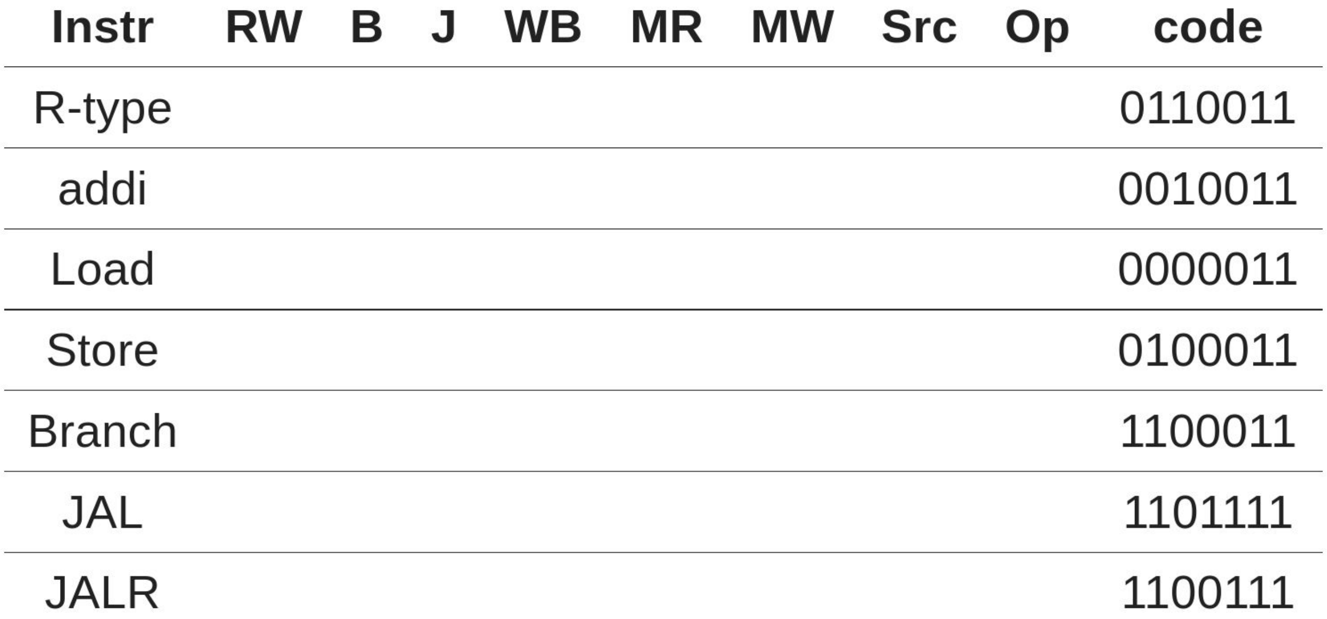
answer.txt



```
PC = 00000000000000000000000001010100
Instrction = 00000000000000000000000000000000
Register
R 1 =          12
R 2 =        1024
R 5 =          -1
R 6 =          45
R 7 =           1
R10 =          55
Data Memory
DataMemory[ 856] =          60
DataMemory[ 864] =           1
DataMemory[ 872] =          60
DataMemory[ 880] =           2
DataMemory[ 888] =          60
DataMemory[ 896] =           3
DataMemory[ 904] =          60
DataMemory[ 912] =           4
DataMemory[ 920] =          60
DataMemory[ 928] =           5
DataMemory[ 936] =          60
DataMemory[ 944] =           6
DataMemory[ 952] =          60
DataMemory[ 960] =           7
DataMemory[ 968] =          60
DataMemory[ 976] =           8
DataMemory[ 984] =          60
DataMemory[ 992] =           9
DataMemory[1000] =          60
DataMemory[1008] =          10
DataMemory[1016] =          12
```

Implement instructions

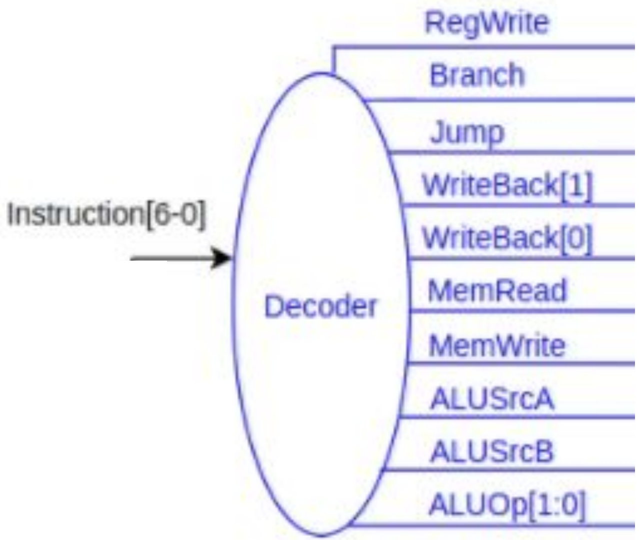
- R-type
  - add
  - slt
- I-type
  - addi
- lw
- sw
- beq
- jal & jalr



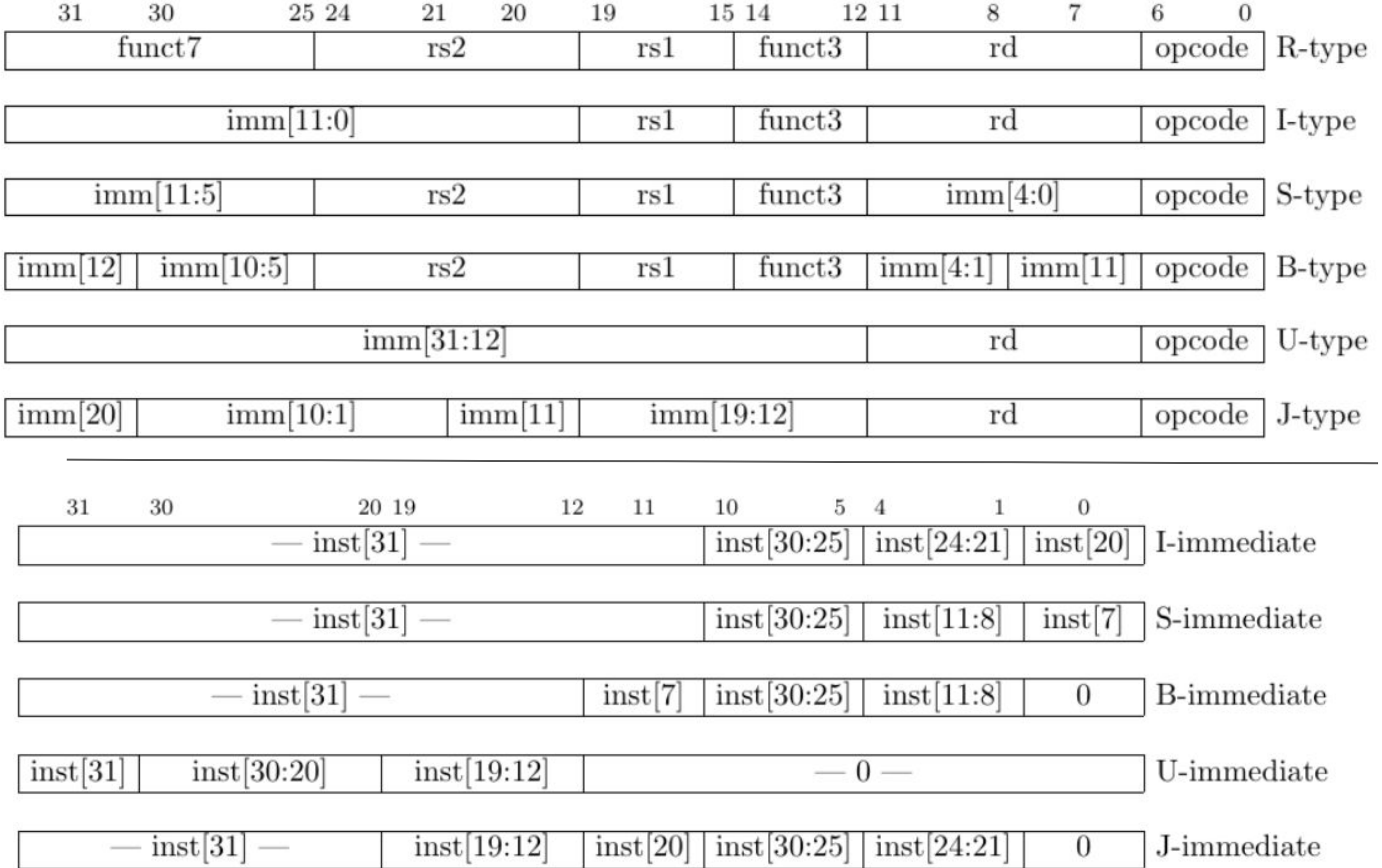
Instr	RW	B	J	WB	MR	MW	Src	Op	code
R-type									0110011
addi									0010011
Load									0000011
Store									0100011
Branch									1100011
JAL									1101111
JALR									1100111

Decoder

- Generate the corresponding control signal according to the instruction
  - RegWrite = ?
  - Branch = ?
  - Jump = ?
  - WriteBack[1] = ?
  - WriteBack[0] = ?
  - MemRead = ?
  - MemWrite = ?
  - ALUSrcA = ?
  - ALUSrcB = ?
  - ALUOp[1:0] = ?



Sign-extend



31 30 25 24 21 20 19 15 14 12 11 8 7 6 0

funct7 | rs2 | rs1 | funct3 | rd | opcode R-type

imm[11:0] | rs1 | funct3 | rd | opcode I-type

imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode S-type

imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode B-type

imm[31:12] | rd | opcode U-type

imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode J-type

31 30 20 19 12 11 10 5 4 1 0

— inst[31] — | inst[30:25] | inst[24:21] | inst[20] I-immediate

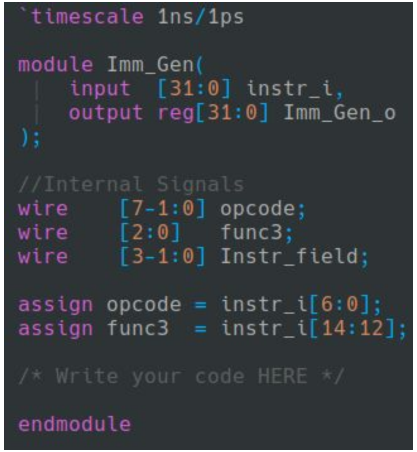
— inst[31] — | inst[30:25] | inst[11:8] | inst[7] S-immediate

— inst[31] — | inst[7] | inst[30:25] | inst[11:8] | 0 B-immediate

inst[31] | inst[30:20] | inst[19:12] | — 0 — U-immediate

— inst[31] — | inst[19:12] | inst[20] | inst[30:25] | inst[24:21] | 0 J-immediate

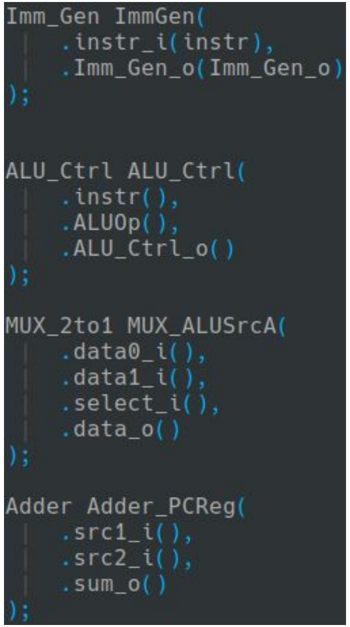
Imm_Gen.v



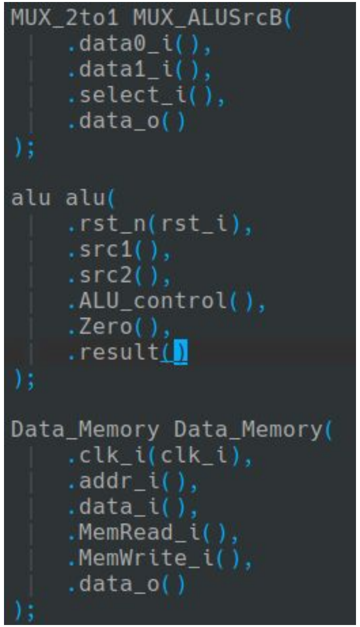
```
`timescale 1ns/1ps

module Imm_Gen(
    input  [31:0] instr_i,
    output reg[31:0] Imm_Gen_o
);

//Internal Signals
wire    [7-1:0] opcode;
wire    [2:0]   func3;
wire    [3-1:0] Instr_field;

assign opcode = instr_i[6:0];
assign func3  = instr_i[14:12];

/* Write your code HERE */

endmodule
```

Simple Single CPU

- Combine all the above parts to build a CPU.
- You need to complete Simple_Single_CPU.v all the parts inputs and output.



```
Imm_Gen ImmGen(
        .instr_i(instr),
        .Imm_Gen_o(Imm_Gen_o)
);


ALU_Ctrl ALU_Ctrl(
        .instr(),
        .ALUOp(),
        .ALU_Ctrl_o()
);

MUX_2to1 MUX_ALUSrcA(
        .data0_i(),
        .data1_i(),
        .select_i(),
        .data_o()
);

Adder Adder_PCReg(
        .src1_i(),
        .src2_i(),
        .sum_o()
);
```

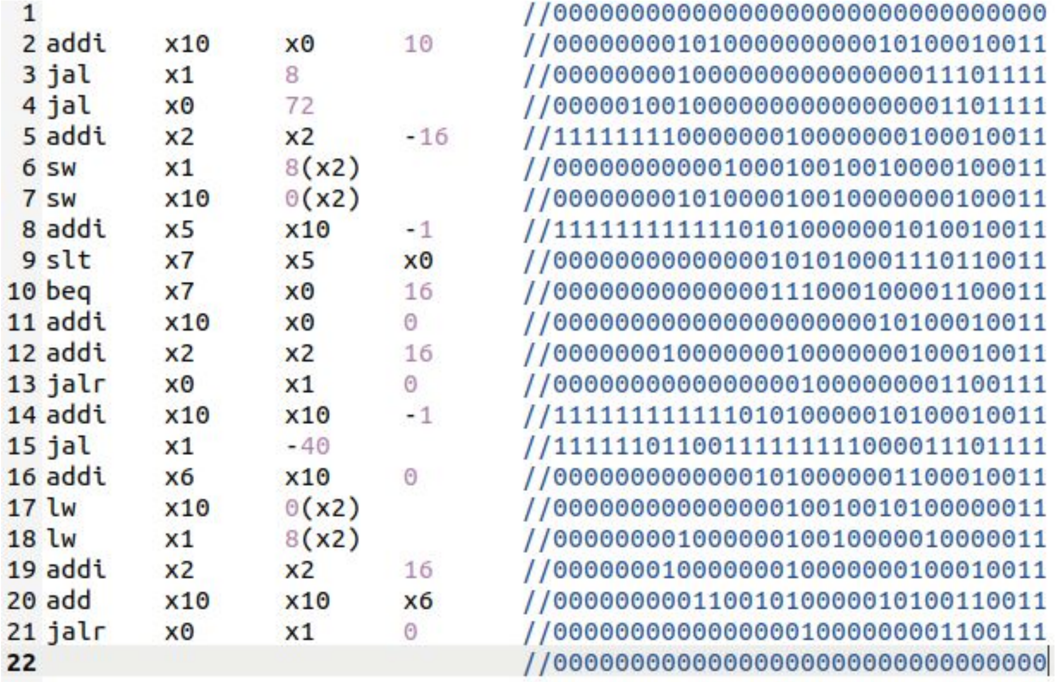


```
MUX_2to1 MUX_ALUSrcB(
        .data0_i(),
        .data1_i(),
        .select_i(),
        .data_o()
);

alu alu(
        .rst_n(rst_i),
        .src1(),
        .src2(),
        .ALU_control(),
        .Zero(),
        .result()
);

Data_Memory Data_Memory(
        .clk_i(clk_i),
        .addr_i(),
        .data_i(),
        .MemRead_i(),
        .MemWrite_i(),
        .data_o()
);
```

....

Test data

- A simple machine code which recursively add numbers from 1 to 10.

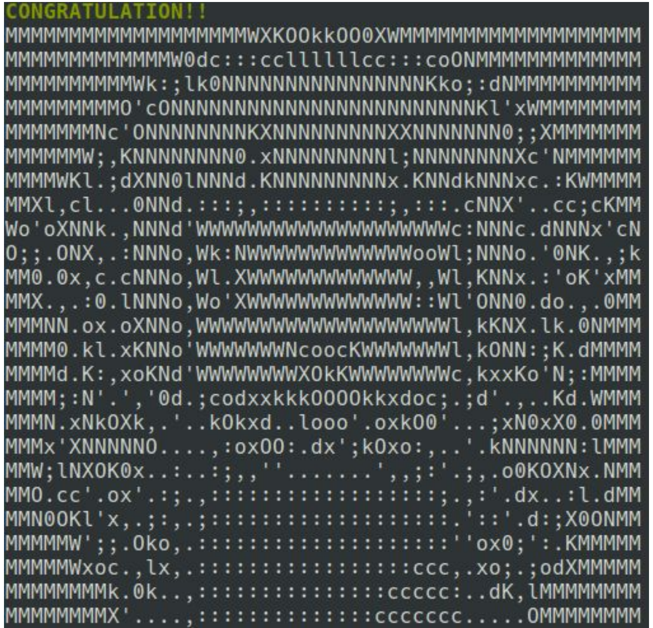


```
                                //00000000000000000000000000000000
addi    x10     x0      10      //00000000101000000000010100010011
jal     x1      8               //00000000100000000000000011101111
jal     x0      72              //00000100100000000000000001101111
addi    x2      x2      -16     //11111111000000010000000100010011
sw      x1      8(x2)           //00000000000100010010010000100011
sw      x10     0(x2)           //00000000101000010010000000100011
addi    x5      x10     -1      //11111111111101010000001010010011
slt     x7      x5      x0      //00000000000000101010001110110011
beq     x7      x0      16      //00000000000000111000100001100011
addi    x10     x0      0       //00000000000000000000010100010011
addi    x2      x2      16      //00000001000000010000000100010011
jalr    x0      x1      0       //00000000000000001000000001100111
addi    x10     x10     -1      //11111111111101010000010100010011
jal     x1      -40             //11111101100111111111000011101111
addi    x6      x10     0       //00000000000001010000001100010011
lw      x10     0(x2)           //00000000000000010010010100000011
lw      x1      8(x2)           //00000000100000010010000010000011
addi    x2      x2      16      //00000001000000010000000100010011
add     x10     x10     x6      //00000000011001010000010100110011
jalr    x0      x1      0       //00000000000000001000000001100111
                                //00000000000000000000000000000000
```

Testbench

- **This script cannot run in Windows**
- Put your.v file in Lab4_release
- $ ./demo

Correct



```
CONGRATULATION!!
MMMMMMMMMMMMMMMMMMMWXKOOkkOOOXWMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMW0dc:::ccllllllcc:::coONMMMMMMMMMMMMM
MMMMMMMMMMWk:;lk0NNNNNNNNNNNNNNNNKko;:dNMMMMMMMMMM
MMMMMMMMMO'cONNNNNNNNNNNNNNNNNNNNNNNKl'xWMMMMMMMMM
MMMMMMMNc'ONNNNNNNNKXNNNNNNNNNXXNNNNNNN0;;XMMMMMMM
MMMMMMW;,KNNNNNNNN0.xNNNNNNNNNl;NNNNNNNNXc'NMMMMMM
MMMMWKl.;dXNN0lNNNd.KNNNNNNNNNx.KNNdkNNNxc.:KWMMMM
MMXl,cl...0NNd.:::;,::::::::::;,:::.cNNX'..cc;cKMM
Wo'oXNNk.,NNNd'WWWWWWWWWWWWWWWWWWWWc:NNNc.dNNNx'cN
O;;.ONX,.:NNNo,Wk:NWWWWWWWWWWWWWooWl;NNNo.'0NK.,;k
MM0.0x,c.cNNNo,Wl.XWWWWWWWWWWWWW,,Wl,KNNx.:'oK'xMM
MMX.,.:0.lNNNo,Wo'XWWWWWWWWWWWWW::Wl'ONN0.do.,.0MM
MMMNN.ox.oXNNo,WWWWWWWWWWWWWWWWWWWWl,kKNX.lk.0NMMM
MMMM0.kl.xKNNo'WWWWWWWNcoocKWWWWWWWl,kONN:;K.dMMMM
MMMMd.K:,xoKNd'WWWWWWWWXOkKWWWWWWWWc,kxxKo'N;:MMMM
MMMM;:N'.','0d.;codxxkkkOOOOkkxdoc;.;d'.,..Kd.WMMM
MMMN.xNkOXk,.'..kOkxd..looo'.oxkO0'...;xN0xX0.0MMM
MMMx'XNNNNNO....,:oxOO:.dx';kOxo:,..'.kNNNNNN:lMMM
MMW;lNXOK0x..:..:;,,''.......',,;:'.;,.o0KOXNx.NMM
MMO.cc'.ox'.:;.,::::::::::::::::::;.,:'.dx..:l.dMM
MMN0OKl'x,.;:,.;:::::::::::::::::::.'::'.d:;X0ONMM
MMMMMW';;.Oko,.::::::::::::::::::::''ox0;':.KMMMMM
MMMMMWxoc.,lx,.:::::::::::::::::ccc,.xo;.;odXMMMMM
MMMMMMMMk.0k..,:::::::::::::::ccccc:..dK,lMMMMMMMM
MMMMMMMMX'....,::::::::::::::ccccccc.....OMMMMMMMM
```

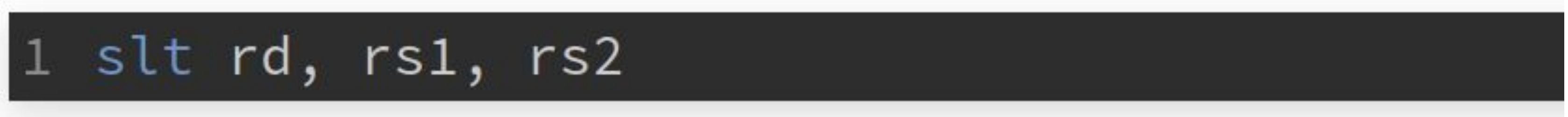
Notice

- Lab4討論區
- Lab4-6分組表

Appendix

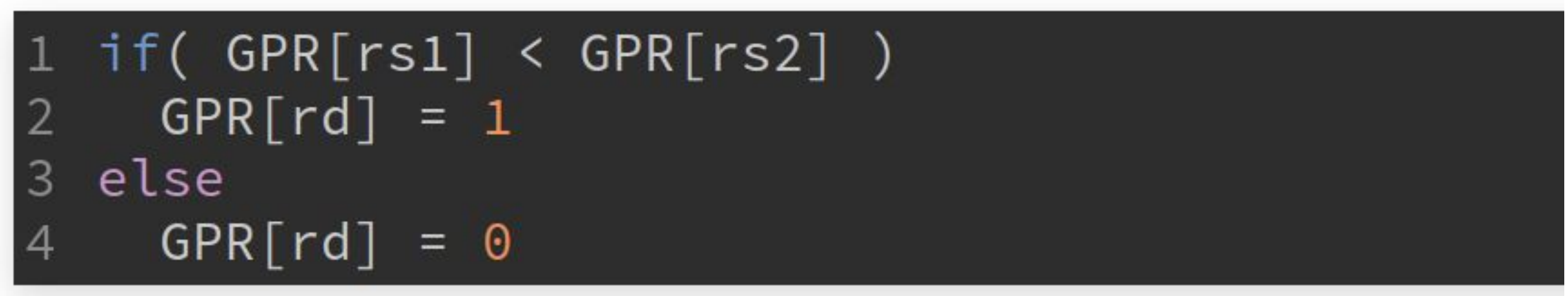
R-type slt

- **Assembly**



```
slt rd, rs1, rs2
```

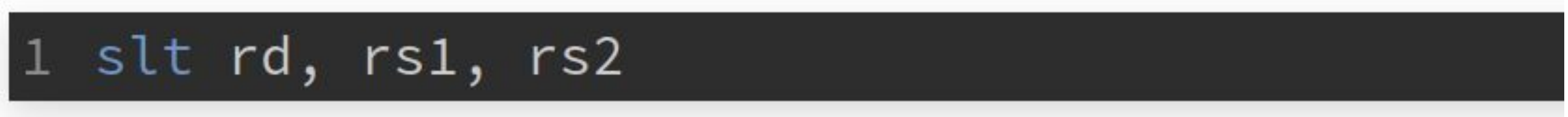
- **Semantics**



```
if( GPR[rs1] < GPR[rs2] )
  GPR[rd] = 1
else
  GPR[rd] = 0
```

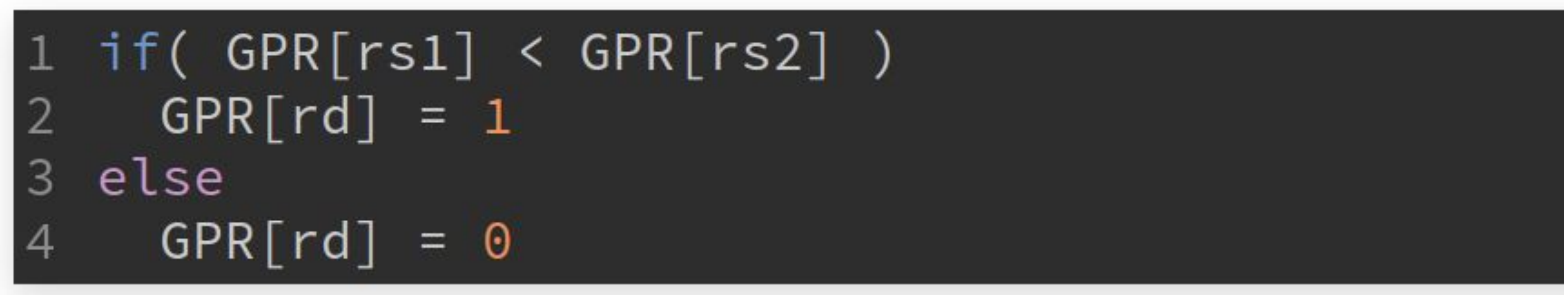
R-type slt

- **Assembly**



```
slt rd, rs1, rs2
```

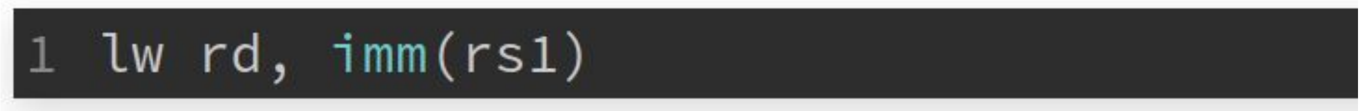
- **Semantics**



```
if( GPR[rs1] < GPR[rs2] )
  GPR[rd] = 1
else
  GPR[rd] = 0
```

Load

- **Assembly**



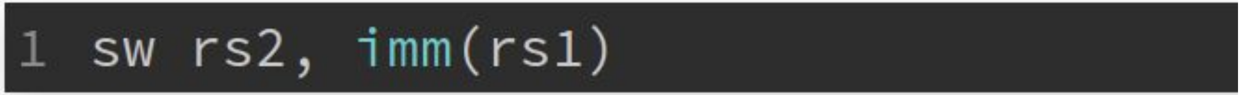
```
lw rd, imm(rs1)
```

- **Semantics**

```
target = GPR[rs1] + sign-extend(imm_{12})
GPR[rd] = DM[target]
```

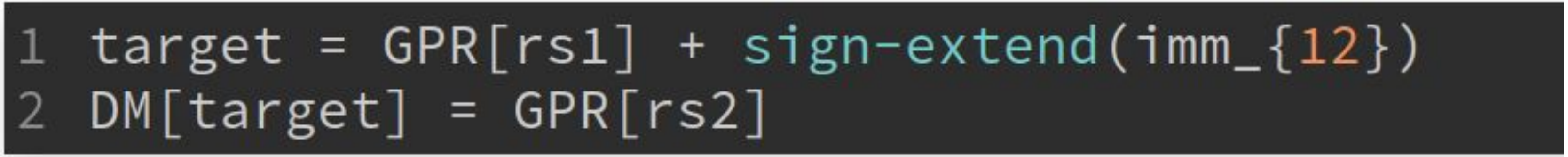
Store

- **Assembly**



```
sw rs2, imm(rs1)
```

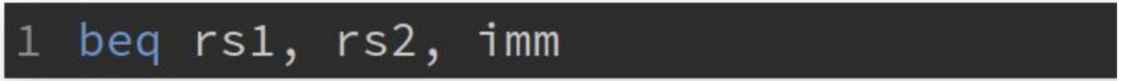
- **Semantics**



```
target = GPR[rs1] + sign-extend(imm_{12})
DM[target] = GPR[rs2]
```

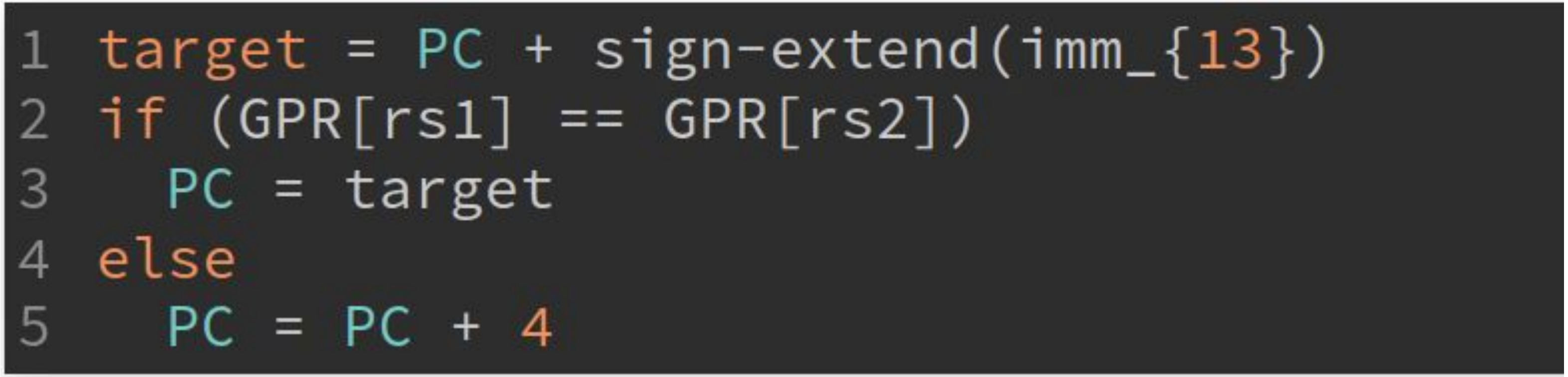
Branch

- **Assembly**



```
beq rs1, rs2, imm
```

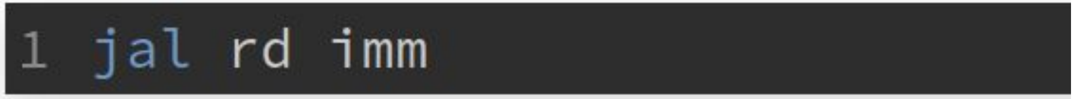
- **Semantics**



```
target = PC + sign-extend(imm_{13})
if (GPR[rs1] == GPR[rs2])
  PC = target
else
  PC = PC + 4
```

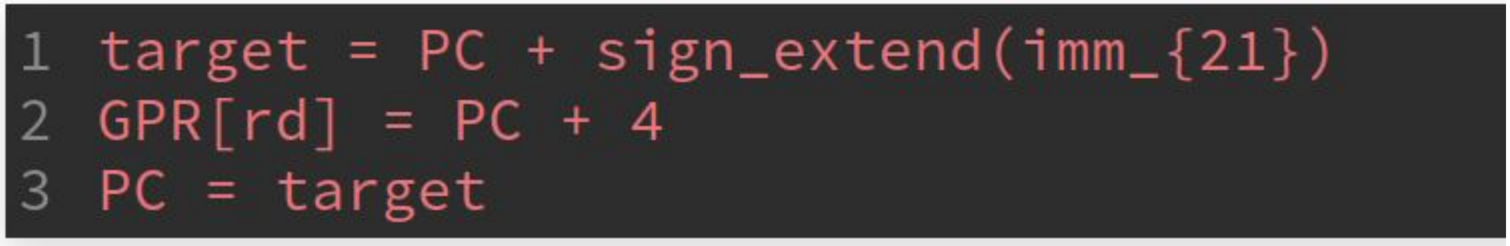
Jump and link

- **Assembly**



```
jal rd imm
```

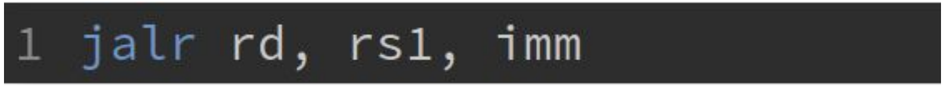
- **Semantics**



```
target = PC + sign_extend(imm_{21})
GPR[rd] = PC + 4
PC = target
```

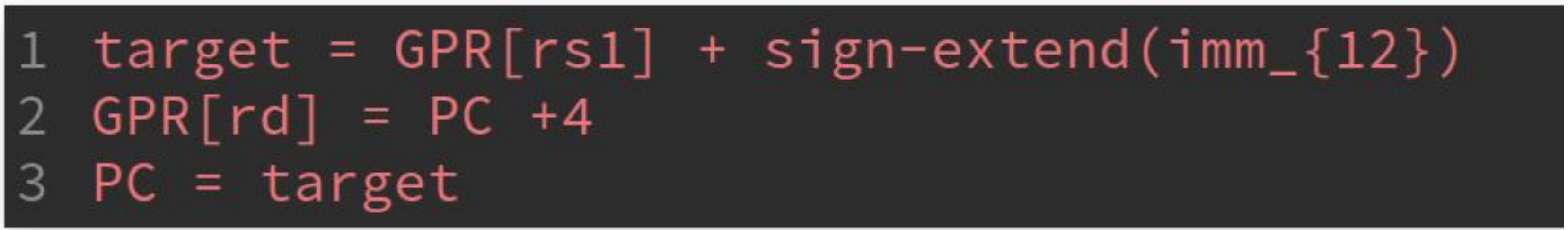
Jump indirect

- **Assembly**



```
jalr rd, rs1, imm
```

- **Semantics**



```
target = GPR[rs1] + sign-extend(imm_{12})
GPR[rd] = PC +4
PC = target
```