154B Discussion 3

January 25th, 2023

Goals

- Assignment 2
  - Datapath.
  - Memory Instructions.
  - Data Memory Interface.
- Decoding a RISC-V instruction.
- RISC vs CISC: examples of X86.
- Week 3 Quiz.

Logistics

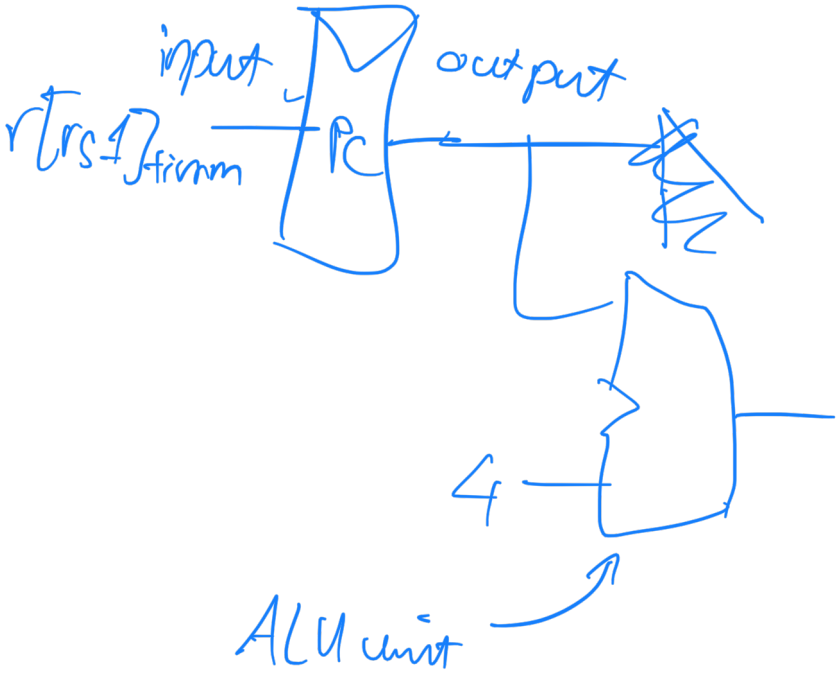
- Assignment 3 released.
  - More boilerplate comments in pipelined/cpu.scala will be added.

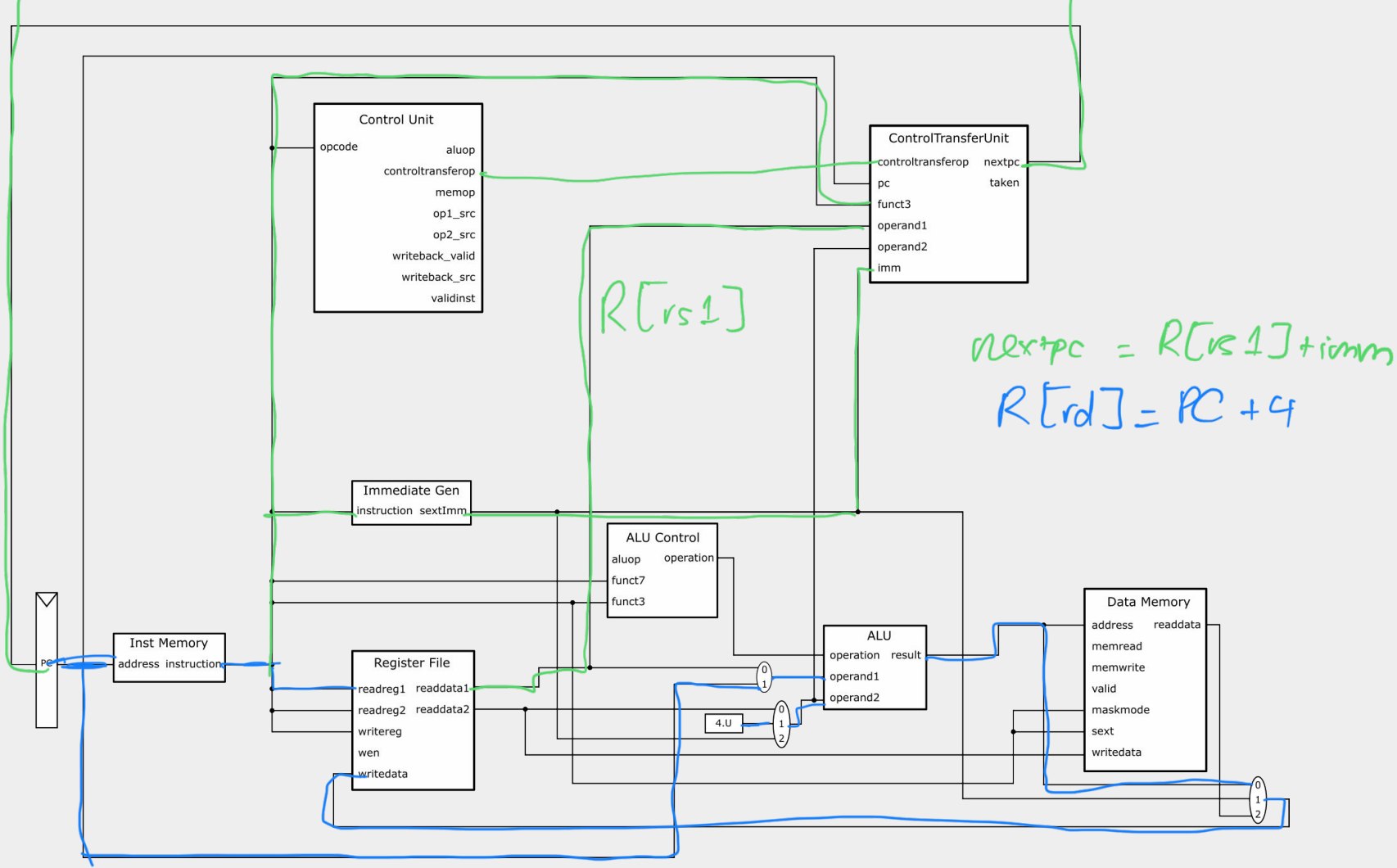
Assignment 2: Datapath for an instruction

- Consider what the instruction does.
  - Does it read from the register file?
  - Does it write to the register file?
  - Does it read from memory?
  - Does it write to memory?
  - Does it affect the next PC?
  - Does it have an immediate value?
- Decide which component does what. E.g., a load instruction,
  - Needs to calculate the effective address.
  - The effective address needs to be sent to the data memory component.
  - The value gotten from the memory system needs to be written to the destination register.

Assignment 2: Datapath for JALR

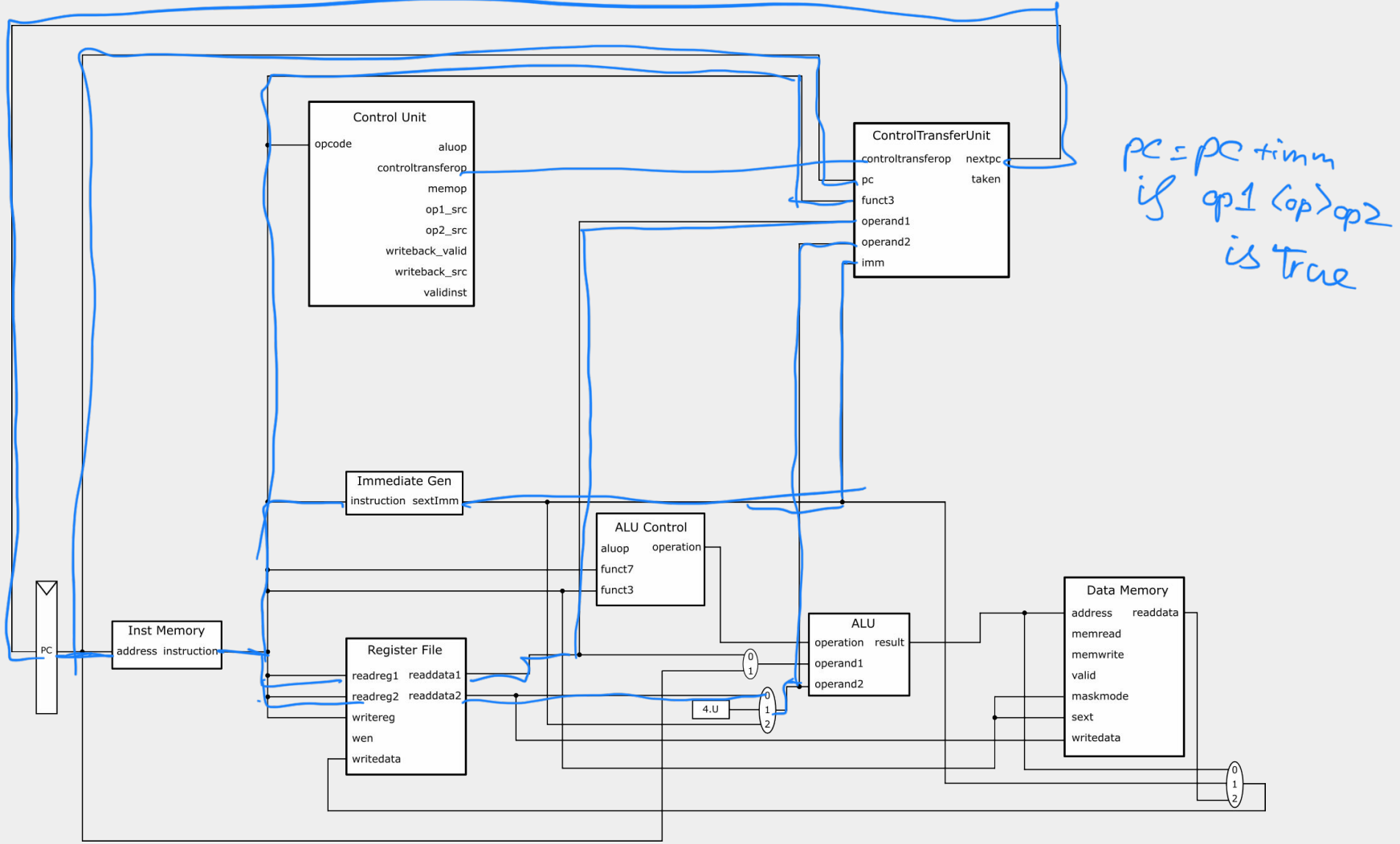
```
pc = R[rs1] + imm
R[rd] = pc + 4
```




Control Unit
opcode
aluop
controltransferop
memop
op1_src
op2_src
writeback_valid
writeback_src
validinst
ControlTransferUnit
controltransferop
nextpc
pc
taken
funct3
operand1
operand2
imm
R[rs1]
nextpc = R[rs1] + imm
R[rd] = PC + 4
Immediate Gen
instruction
sextImm
ALU Control
aluop
operation
funct7
funct3
Inst Memory
address
instruction
Register File
readreg1
readdata1
readreg2
readdata2
writereg
wen
writedata
4.U
ALU
operation
result
operand1
operand2
Data Memory
address
readdata
memread
memwrite
valid
maskmode
sext
writedata

Assignment 2: Datapath for Branches (B-type insts)

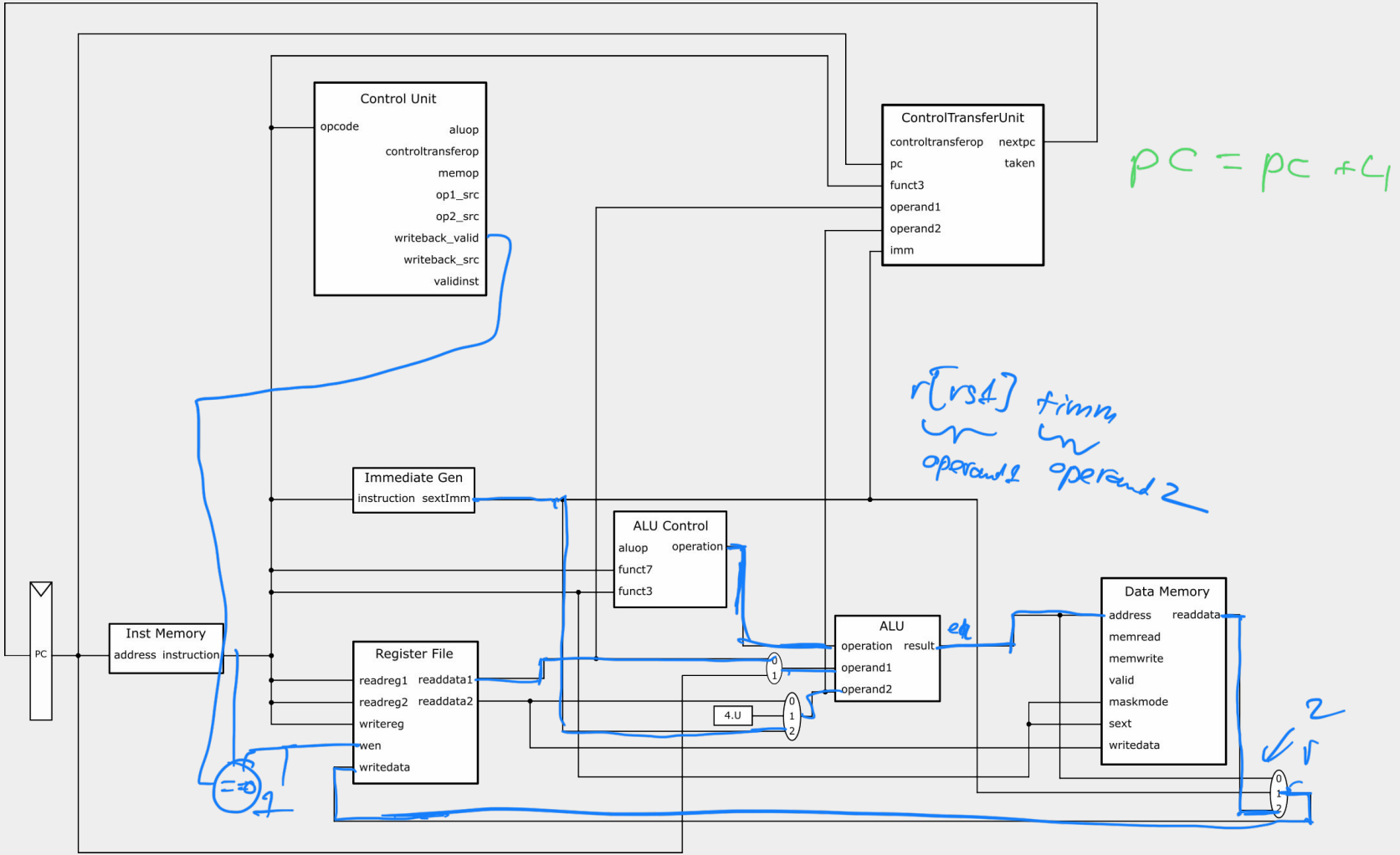
```
if (R[rs1] <op> R[rs2])
  pc = pc + immediate
else
  pc = pc + 4
```


Control Unit
opcode
aluop
controltransferop
memop
op1_src
op2_src
writeback_valid
writeback_src
validinst
ControlTransferUnit
controltransferop
nextpc
pc
taken
funct3
operand1
operand2
imm
pc = pc + imm
if op1 <op> op2
is true
Immediate Gen
instruction
sextImm
ALU Control
aluop
operation
funct7
funct3
Data Memory
address
readdata
memread
memwrite
valid
maskmode
sext
writedata
ALU
operation
result
operand1
operand2
PC
Inst Memory
address
instruction
Register File
readreg1
readdata1
readreg2
readdata2
writereg
wen
writedata
4.U
0
1
2

Assignment 2: Datapath for Load instructions (I-type inst)

```
R[rd] = M[R[rs1] + immediate]
```

effective address


Control Unit
opcode
aluop
controltransferop
memop
op1_src
op2_src
writeback_valid
writeback_src
validinst
ControlTransferUnit
controltransferop
nextpc
pc
taken
funct3
operand1
operand2
imm
pc = pc +4
r[rs1]
imm
operand1
operand2
Immediate Gen
instruction
sextImm
ALU Control
aluop
operation
funct7
funct3
Data Memory
address
readdata
memread
memwrite
valid
maskmode
sext
writedata
ALU
operation
result
operand1
operand2
PC
Inst Memory
address
instruction
Register File
readreg1
readdata1
readreg2
readdata2
writereg
wen
writedata
4.U
0
1
2
=0

RISC-V Memory Instructions

opcode

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	110	rd	0000011	LWU
imm[11:0]		rs1	011	rd	0000011	LD
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	SD

RISC-V Memory Instructions: Loads

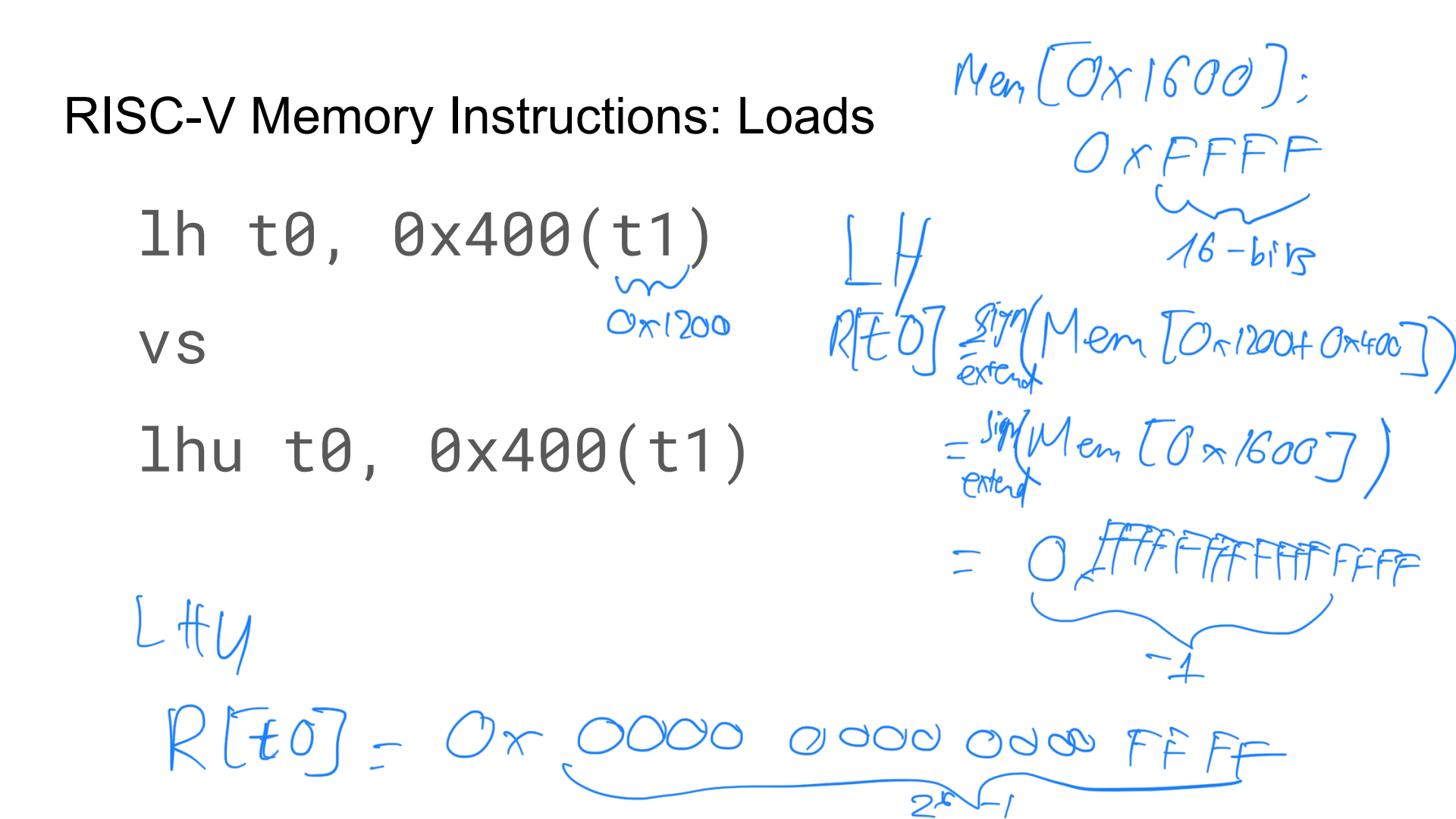
- B, H, W, and D mean Byte (8 bits), Half-word (16 bits), Word (32 bits), and Doubleword (64 bits).
- U-suffix means treating the loaded value as an unsigned integer.
- Without the U-suffix, the loaded value will be treated as a signed integer.
- The value read from the memory might be shorter than 64 bits long; however, the value written to the destination register must be a 64-bit value.
- Implications,
  - A load instruction updates all 64 bits of the destination register (this is very different from x86 load(mov) instructions).
  - The loaded value must be sign-extended if it is treated as a signed integer.

RISC-V Memory Instructions: Loads

- E.g., LH means loading from memory a 16-bit value, which will be signed-extended to a 64-bit value before written back to the destination register.
- E.g., LHU means loading from memory a 16-bit value, which will **not** be signed-extended to a 64-bit value before written back to the destination register.
- In DINOCPU, if you set dmem.io.sext to true, the output dmem.io.readdata will contain the loaded data sign-extended to its 64-bit representation.

RISC-V Memory Instructions: Loads

```
lh t0, 0x400(t1)
vs
lhu t0, 0x400(t1)
```



RISC-V Memory Instructions: Stores

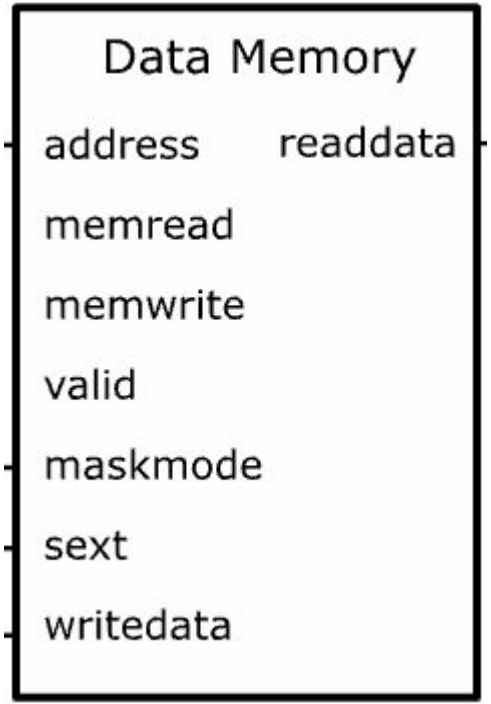
- No sign-extension involved.
- E.g., SH will write the 16 least significant bits of the source register to the memory location specified by the effective address.

RISC-V Memory Instructions

- Load and store instructions are the only instructions that interact with memory. [1]
- Arithmetic instructions do not load from memory. [2]
- As a side note, due to [1] and [2], RISC-V, falls under the category of “load-store architecture” (as opposed to “register-memory architecture”, e.g. X86).

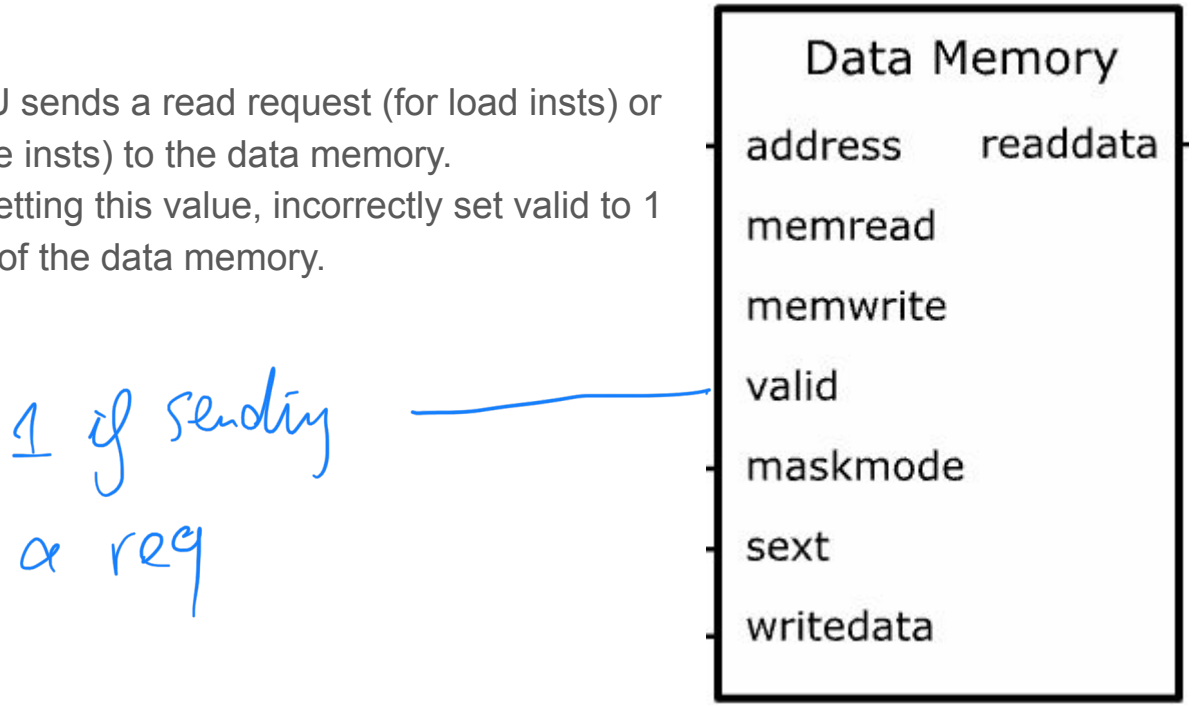
DINOCPU Data Memory Interface

- How the CPU interacts with Data Memory?
  - CPU sends a **request**.
  - Data Memory received the request.
  - Data Memory sends a response.
  - CPU receives a **response**.
- For assignment 2 and assignment 3, we are using combinational memory model, meaning CPU sends a request and receives the response in the same cycle.



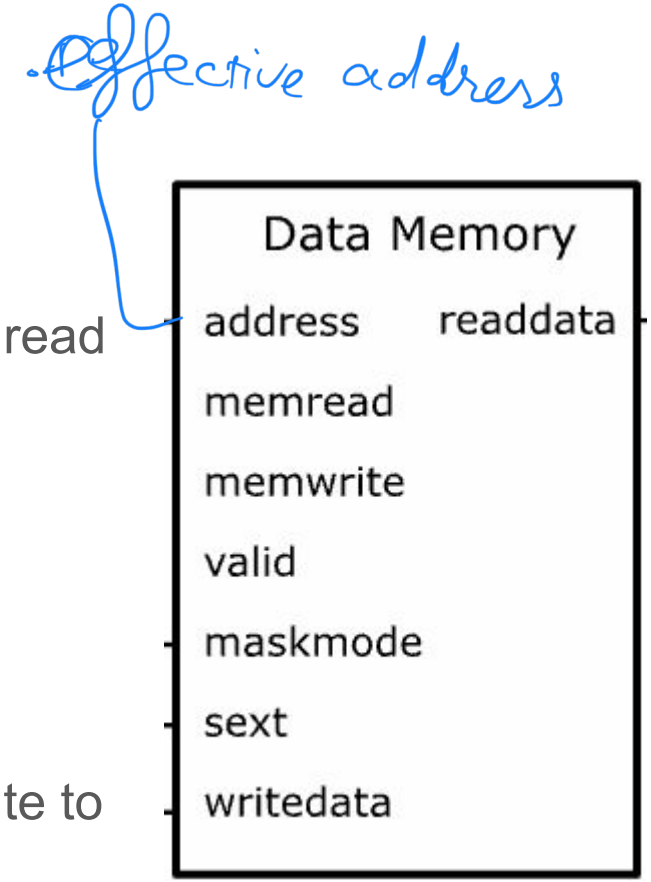
DINOCPU Data Memory Interface

- valid:
  - when valid = 1: the CPU sends a read request (for load insts) or a write request (for store insts) to the data memory.
  - must be careful when setting this value, incorrectly set valid to 1 might change the state of the data memory.



DINOCPU Data Memory Interface

- memread: the request is a read request.
- memwrite: the request is a write request.
- maskmode (used for both read/write): size of read data or write data.
  - 0.U -> byte (8 bits) LB | SB
  - 1.U -> halfword (16 bits) LH | SH
  - 2.U -> word (32 bits) LW | SW
  - 3.U -> doubleword (64 bits) LD | SD
- sext (used for read requests):
  - If set to 1, the value loaded from memory will be sign-extended to its 64-bit value.
- writedata (used in write requests): what to write to memory.



Decoding a RISC-V Instruction

- Read the opcode first (always at bit 6 -> bit 0).
- The opcode decides the instruction format.
- Use the instruction format to decode the rs1, rs2, rd, funct3, funct7, imm, etc. of the instruction.

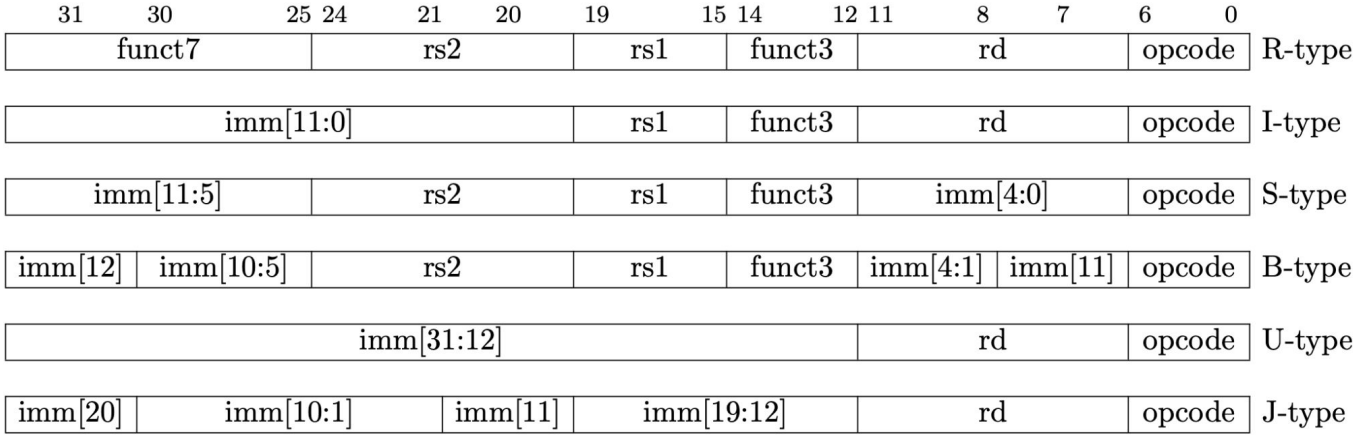


Figure 2.3: RISC-V base instruction formats showing immediate variants.

RISC vs CISC: examples from X86

(Screenshot from https://www.felixcloutier.com/x86/add)

* Multiplication instruction.

→ In X86, the MUL r64 instruction performs 64 bits unsigned integers multiplication. The result is 128 bits long, and the 64 MSB are written to RDX, while the 64 LSB are written to RAX.

→ In RISC-V, it costs 2 instructions to produce the same effect.

The MUL inst produces the 64 LSB of the multiplication.

MULHU inst produces the 64 MSB of the multiplication.

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	MI	Valid	Valid	Add *imm8* to *r/m8*.
REX + 80 /0 *ib*	ADD *r/m8*, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m8*.
81 /0 *iw*	ADD *r/m16, imm16*	MI	Valid	Valid	Add *imm16* to *r/m16*.
81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64*.
83 /0 *ib*	ADD *r/m16, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m16*.
83 /0 *ib*	ADD *r/m32, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m32*.
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64*.
00 /*r*	ADD *r/m8, r8*	MR	Valid	Valid	Add *r8* to *r/m8*.
REX + 00 /*r*	ADD *r/m8*, r8**	MR	Valid	N.E.	Add *r8* to *r/m8*.
01 /*r*	ADD *r/m16, r16*	MR	Valid	Valid	Add *r16* to *r/m16*.
01 /*r*	ADD *r/m32, r32*	MR	Valid	Valid	Add *r32* to *r/m32*.
REX.W + 01 /*r*	ADD *r/m64, r64*	MR	Valid	N.E.	Add *r64* to *r/m64*.
02 /*r*	ADD *r8, r/m8*	RM	Valid	Valid	Add *r/m8* to *r8*.
REX + 02 /*r*	ADD *r8*, r/m8**	RM	Valid	N.E.	Add *r/m8* to *r8*.
03 /*r*	ADD *r16, r/m16*	RM	Valid	Valid	Add *r/m16* to *r16*.
03 /*r*	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.
REX.W + 03 /*r*	ADD *r64, r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64*.

x86 has instructions to add a value from part of 1 register to a value in memory.

The equivalent in RISC-V is to load the value from memory to a register, then add 2 regs

Week 3 Quiz

1.2 1.5 1.7

$\sqrt[3]{1.2 \times 1.5 \times 1.7}$

- Question 3: Use geometric means.
- Question 7:
  - How the hardware is implemented is specified by the **microarchitecture**, not the ISA/architecture.
  - The number of **logical** registers is defined by the ISA.
  - The number of **physical** registers is defined by the microarchitecture.
  - However, when we say “number of registers”, we usually refer to the number of **logical** registers.
  - Virtual memory and size of registers are defined by ISA.
- Question 13: official RISC-V ISA extensions are not for customers to add customized instructions.

Week 3 Quiz

Decoding `11111111000101111110001011100011`

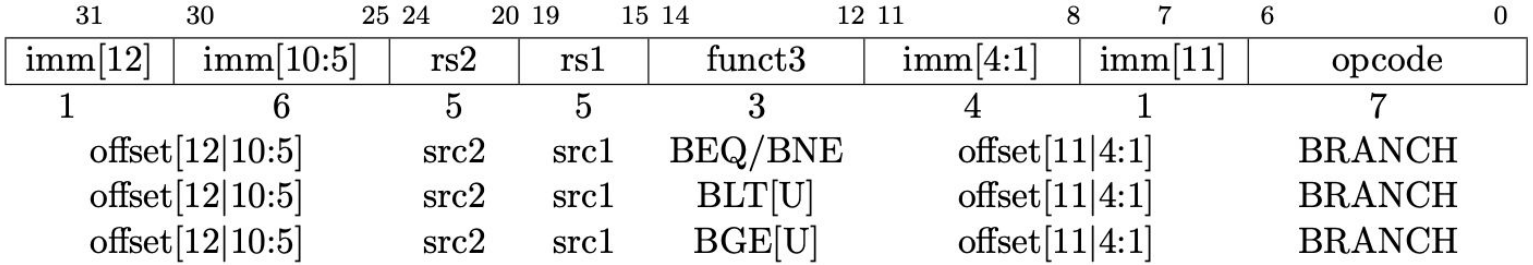
Week 3 Quiz

`11111111000101111111000101`**`1100011`**

opcode
-> B-type

Conditional Branches

All branch instructions use the B-type instruction format. The 12-bit B-immediate encodes signed offsets in multiples of 2 bytes. The offset is sign-extended and added to the address of the branch instruction to give the target address. The conditional branch range is ±4 KiB.



31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		src2	src1	BEQ/BNE	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BLT[U]	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BGE[U]	offset[11|4:1]		BRANCH

Week 3 Quiz

`11111111000101111111000101`**`1100011`**

opcode
-> B-type

There's no imm[0] because this bit is always 0. Why? the shortest RISC-V instruction length is 16 bits (2 bytes), and RISC-V instruction lengths are always a multiple of 2 bytes.

Conditional Branches

All branch instructions use the B-type instruction format. The 12-bit B-immediate encodes signed offsets in multiples of 2 bytes. The offset is sign-extended and added to the address of the branch instruction to give the target address. The conditional branch range is ±4 KiB.

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		src2	src1	BEQ/BNE	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BLT[U]	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BGE[U]	offset[11|4:1]		BRANCH

Week 3 Quiz

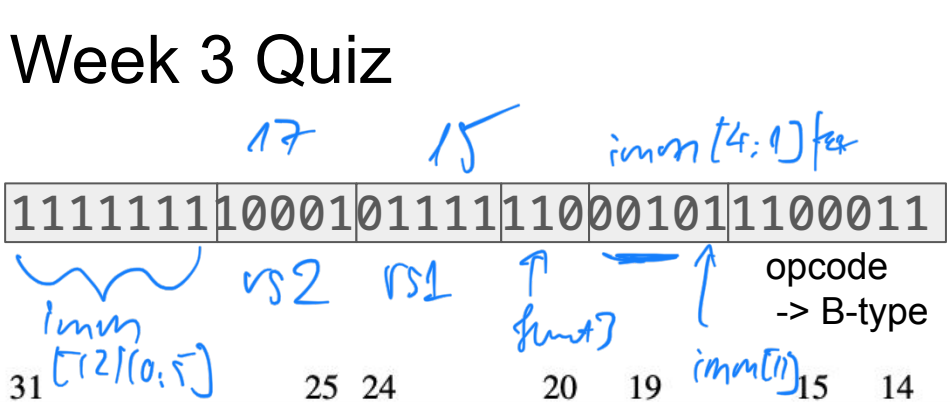
`11111111000101111111000101`**`1100011`**

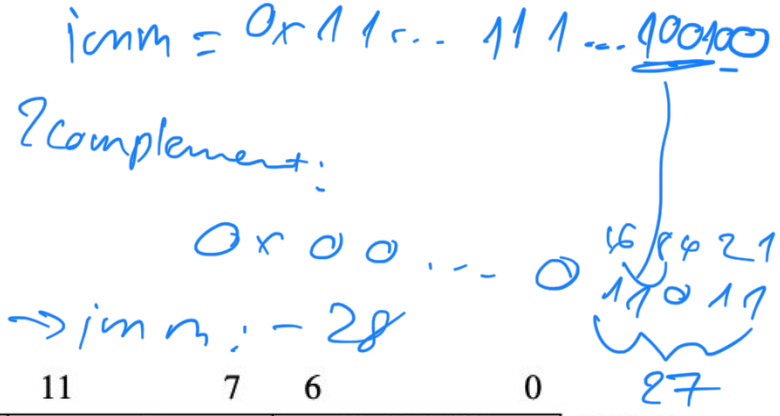
opcode
-> B-type

funct3

31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu

Week 3 Quiz





imm[31:12]				rd	0110111 U lui
imm[31:12]				rd	0010111 U auipc
imm[20|10:1|11|19:12]				rd	1101111 J jal
imm[11:0]		rs1	000	rd	1100111 I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011 B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011 B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011 B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011 B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011 B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011 B bgeu

Week 3 Quiz

- The immediate is treated as a signed integer.
- The U in BLTU means the source register values are treated as unsigned integers.

```
10574:      fe442883                lw      a7,-28(s0)
10578:      01178313                addi    t1,a5,17
1057c:      00f83023                sd      a5,0(a6)
10580:      0066b023                sd      t1,0(a3)
10584:      00073023                sd      zero,0(a4)
10588:      0785                    addi    a5,a5,1
1058a:      0821                    addi    a6,a6,8
1058c:      06a1                    addi    a3,a3,8
1058e:      0721                    addi    a4,a4,8
10590:      ff17e2e3                bltu    a5,a7,10574 <main+0x4e>
```

Week 3 Quiz

Q5: new: 1.5x freq → $freq_{new} = 1.5\, freq_{old}$

1.5x #insts → $\#insts_{new} = 1.5 \times \#insts_{old}$

3 / 1.7 CPI → $\frac{CPI_{new}}{CPI_{old}} = \frac{1.7}{3}$

$$speedup = \frac{t_{old}}{t_{new}} = \frac{\frac{1}{freq_{old}} \times CPI_{old} \times \#insts_{old}}{\frac{1}{freq_{new}} \times CPI_{new} \times \#insts_{new}}$$