ECE M116C / CS 151B: Week 1 Section

Justin Feng

Who am I?

ELECTRICAL & COMPUTER ENGINEERING

UNIVERSITY *of* WASHINGTON

UCLA Samueli Electrical & Computer Engineering

UCLA

My Research…

- **Lab:**
  - SsysArch Lab, advised by Professor Nader Sehatbakhsh
- **Research:**
  - Security
  - Embedded Systems
  - Communications
  - Computer Architecture
  - *Side Channels: large portion of my work*

UCLA

My Research…

- **Research Project examples:**
  - Communications
  - Fingerprinting
  - VR Security
  - *General topic: beneficial uses of side channels*

Contact Info

- **Main:** DM on Campuswire
- **Email:** jfeng10@g.ucla.edu

What about you all?

- Name, year, department, and favorite song / recent song you like

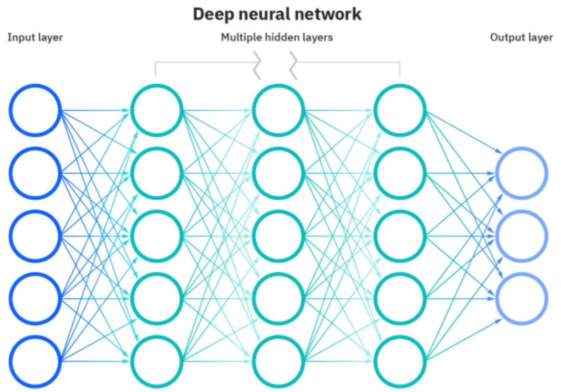
What to Expect from Section?

UCLA

Why is Computer Architecture Important?

Applications

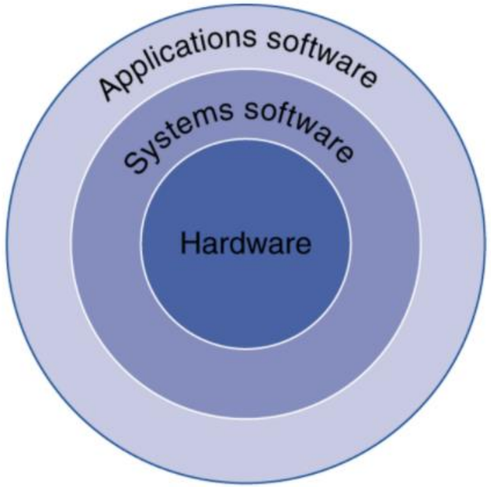
- Your laptops/smartphones
- Enabling machine learning
- Video games
- And more…



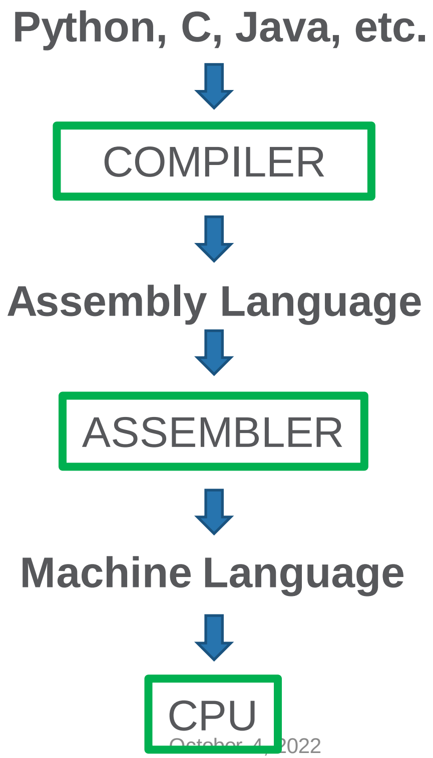
UCLA

What is a Computer?

Computer Abstraction



Level of Program Code



Turing Machine

https://youtu.be/dNRDvLACg5Q?t=67

- **Turing Machine:** the foundation of modern-day computers
  - Manipulating sets of 1’s and 0’s

UCLA

Performance

UCLA

Amdahl’s Law

- Amdahl’s law: our performance is only as good as our slowest part (sequential part)
- BUT, not just about parallelism and the core itself – memory, clock speed, branch prediction all matters!
- Processors:
  - **CPU:** general purpose
  - **Domain specific computing/accelerators:** can be faster than CPU at a specific application

$$speed\ up = \frac{1}{(1-p) + \frac{p}{s}}$$

Amdahl’s Law Example

Ex) If 80% of a program is parallelizable, what is the speedup achievable? (say number of cores changes from 5 to 8)

Answer)

p=0.8 (80% of program is parallelizable), with p=percentage of parallelizable content

s: s=factor of improvement. Dependent on change in number of cores, for instance. Let’s say we increase the number of cores from 5 to 8 (1.6 times as fast), so s=1.6

Speed up = $\frac{1}{(1-0.8)+\frac{0.8}{1.6}}$ = 1.43x speedup

$$speed\ up = \frac{1}{(1-p)+\frac{p}{s}}$$

RISC-V Assembly

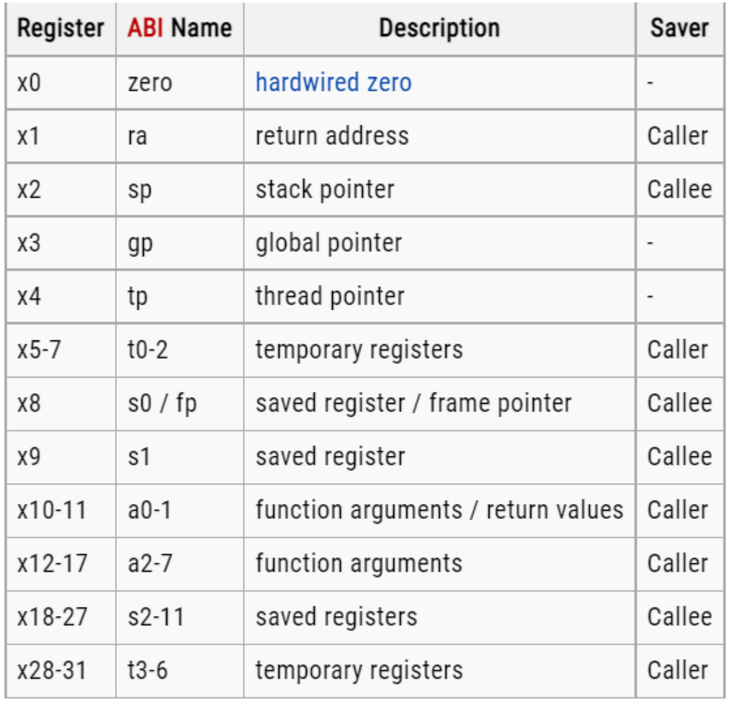
UCLA

RISC vs CISC

- **RISC (Reduced Instruction Set Computer):**
  - ⬇ cycles per instruction, ⬆ total instructions
  - Ex) MIPS, RISC V, ARM
- **CISC (Complex Instruction Set Computer):**
  - ⬇ total instructions, ⬆ cycles per instruction
  - Ex) Intel x86

UCLA

Registers



Register	ABI Name	Description	Saver
x0	zero	hardwired zero	-
x1	ra	return address	Caller
x2	sp	stack pointer	Callee
x3	gp	global pointer	-
x4	tp	thread pointer	-
x5-7	t0-2	temporary registers	Caller
x8	s0 / fp	saved register / frame pointer	Callee
x9	s1	saved register	Callee
x10-11	a0-1	function arguments / return values	Caller
x12-17	a2-7	function arguments	Caller
x18-27	s2-11	saved registers	Callee
x28-31	t3-6	temporary registers	Caller

Note:
Caller Saved: in subprocess, don't need to restore value (caller's responsibility to save value – on stack for instance)
Callee Saved: in subprocess, need to restore value (caller expects value to be saved)

General Syntax

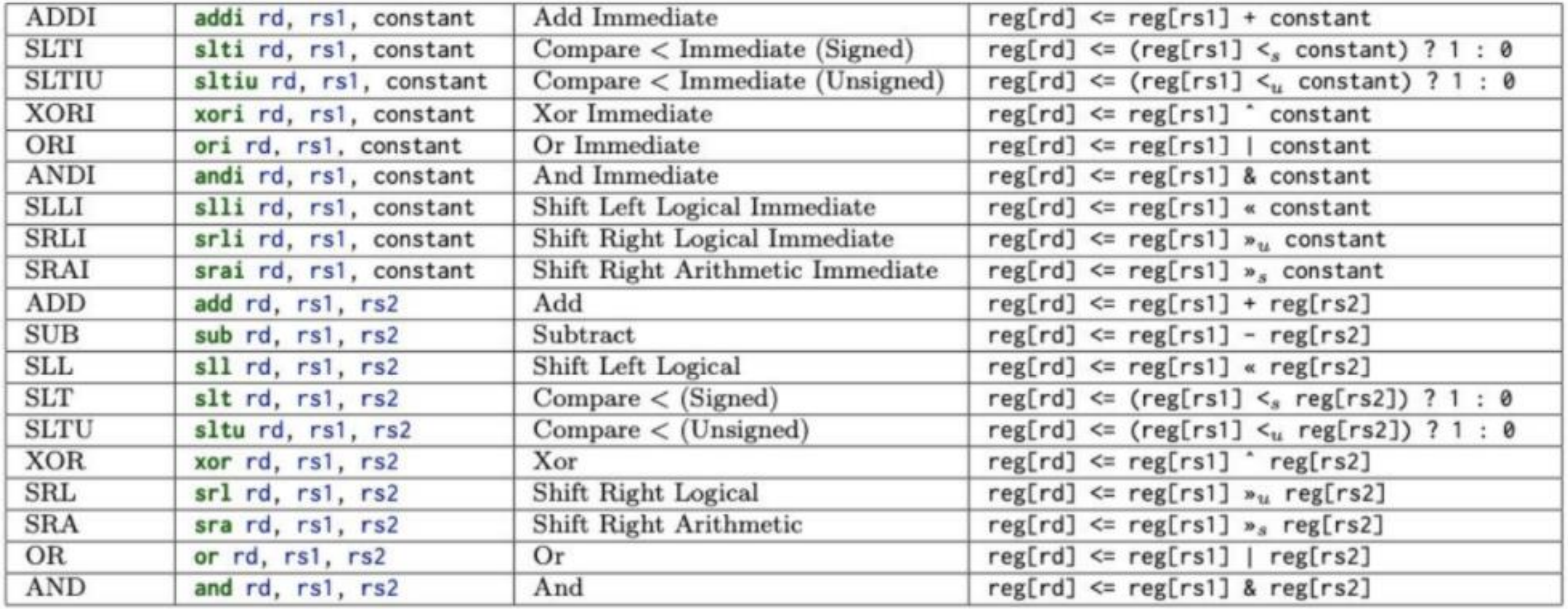
Arithmetic:	add rd, rs1, rs2
	sub rd, rs1, rs2
Logical:	and rd, rs1, rs2
Branch:	blt rs1, rs2, label
Load/Store	lb rd, imm(rs1)
	sb rs2, imm(rs1)
Jumping/Linking	jal rd, label

Note:

What is “label”: an offset of instruction addresses

UCLA

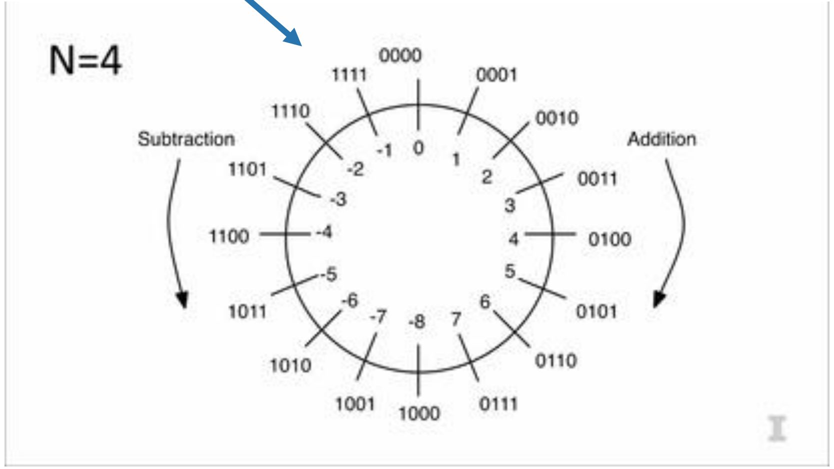
Syntax Examples



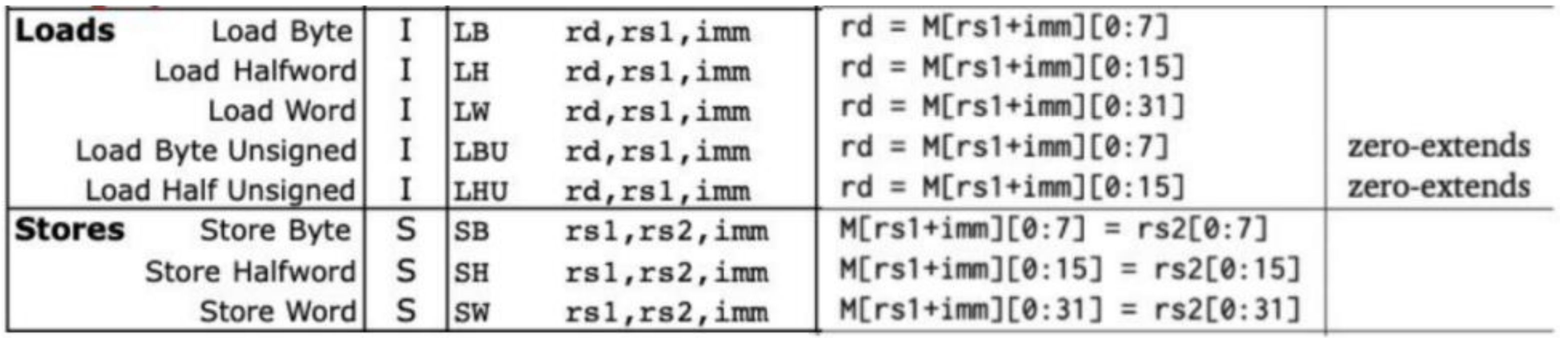
ADDI	addi rd, rs1, constant	Add Immediate	reg[rd] <= reg[rs1] + constant
SLTI	slti rd, rs1, constant	Compare < Immediate (Signed)	reg[rd] <= (reg[rs1] $<_s$ constant) ? 1 : 0
SLTIU	sltiu rd, rs1, constant	Compare < Immediate (Unsigned)	reg[rd] <= (reg[rs1] $<_u$ constant) ? 1 : 0
XORI	xori rd, rs1, constant	Xor Immediate	reg[rd] <= reg[rs1] ^ constant
ORI	ori rd, rs1, constant	Or Immediate	reg[rd] <= reg[rs1] | constant
ANDI	andi rd, rs1, constant	And Immediate	reg[rd] <= reg[rs1] & constant
SLLI	slli rd, rs1, constant	Shift Left Logical Immediate	reg[rd] <= reg[rs1] $\ll$ constant
SRLI	srli rd, rs1, constant	Shift Right Logical Immediate	reg[rd] <= reg[rs1] $\gg_u$ constant
SRAI	srai rd, rs1, constant	Shift Right Arithmetic Immediate	reg[rd] <= reg[rs1] $\gg_s$ constant
ADD	add rd, rs1, rs2	Add	reg[rd] <= reg[rs1] + reg[rs2]
SUB	sub rd, rs1, rs2	Subtract	reg[rd] <= reg[rs1] - reg[rs2]
SLL	sll rd, rs1, rs2	Shift Left Logical	reg[rd] <= reg[rs1] $\ll$ reg[rs2]
SLT	slt rd, rs1, rs2	Compare < (Signed)	reg[rd] <= (reg[rs1] $<_s$ reg[rs2]) ? 1 : 0
SLTU	sltu rd, rs1, rs2	Compare < (Unsigned)	reg[rd] <= (reg[rs1] $<_u$ reg[rs2]) ? 1 : 0
XOR	xor rd, rs1, rs2	Xor	reg[rd] <= reg[rs1] ^ reg[rs2]
SRL	srl rd, rs1, rs2	Shift Right Logical	reg[rd] <= reg[rs1] $\gg_u$ reg[rs2]
SRA	sra rd, rs1, rs2	Shift Right Arithmetic	reg[rd] <= reg[rs1] $\gg_s$ reg[rs2]
OR	or rd, rs1, rs2	Or	reg[rd] <= reg[rs1] | reg[rs2]
AND	and rd, rs1, rs2	And	reg[rd] <= reg[rs1] & reg[rs2]

Logical vs Arithmetic Shift?

- **Logical Shift:** shift in zeros
- **Arithmetic Shift:** keeps sign (remember: 2’s complement), shifting in a one or zero if needed

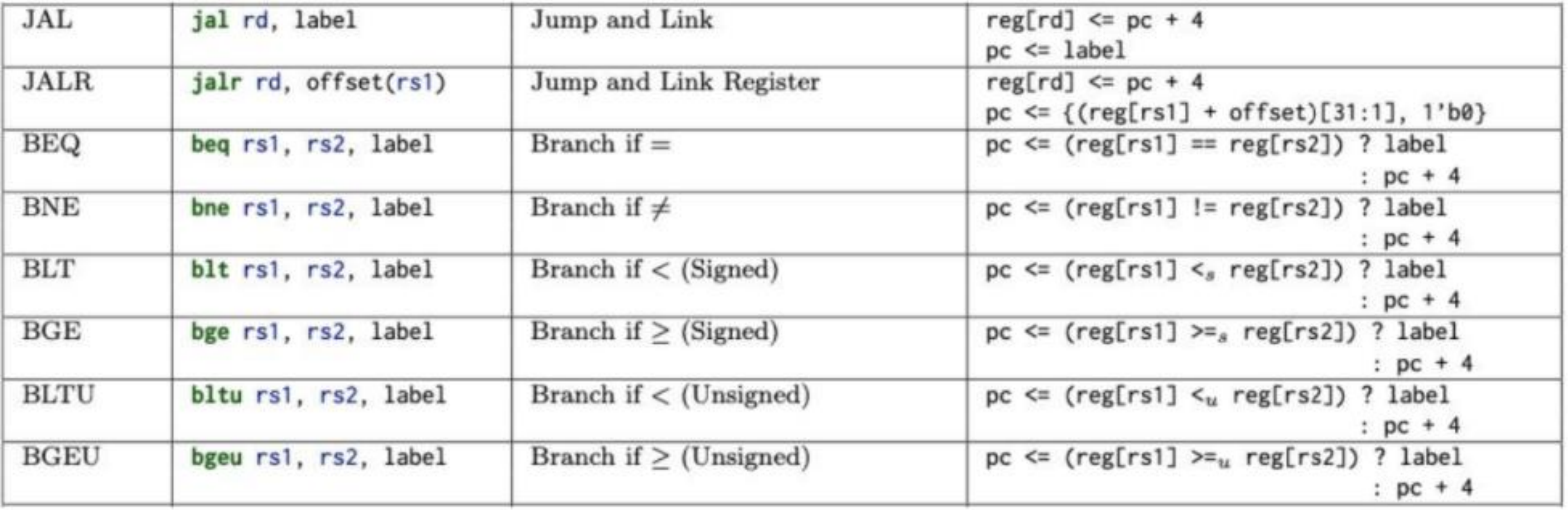


Syntax Examples



Loads	Load Byte	I	LB	rd,rs1,imm	rd = M[rs1+imm][0:7]	
	Load Halfword	I	LH	rd,rs1,imm	rd = M[rs1+imm][0:15]	
	Load Word	I	LW	rd,rs1,imm	rd = M[rs1+imm][0:31]	
	Load Byte Unsigned	I	LBU	rd,rs1,imm	rd = M[rs1+imm][0:7]	zero-extends
	Load Half Unsigned	I	LHU	rd,rs1,imm	rd = M[rs1+imm][0:15]	zero-extends
Stores	Store Byte	S	SB	rs1,rs2,imm	M[rs1+imm][0:7] = rs2[0:7]	
	Store Halfword	S	SH	rs1,rs2,imm	M[rs1+imm][0:15] = rs2[0:15]	
	Store Word	S	SW	rs1,rs2,imm	M[rs1+imm][0:31] = rs2[0:31]	

Syntax Examples



JAL	jal rd, label	Jump and Link	reg[rd] <= pc + 4 pc <= label
JALR	jalr rd, offset(rs1)	Jump and Link Register	reg[rd] <= pc + 4 pc <= {(reg[rs1] + offset)[31:1], 1'b0}
BEQ	beq rs1, rs2, label	Branch if =	pc <= (reg[rs1] == reg[rs2]) ? label : pc + 4
BNE	bne rs1, rs2, label	Branch if ≠	pc <= (reg[rs1] != reg[rs2]) ? label : pc + 4
BLT	blt rs1, rs2, label	Branch if < (Signed)	pc <= (reg[rs1] $<_s$ reg[rs2]) ? label : pc + 4
BGE	bge rs1, rs2, label	Branch if ≥ (Signed)	pc <= (reg[rs1] $>=_s$ reg[rs2]) ? label : pc + 4
BLTU	bltu rs1, rs2, label	Branch if < (Unsigned)	pc <= (reg[rs1] $<_u$ reg[rs2]) ? label : pc + 4
BGEU	bgeu rs1, rs2, label	Branch if ≥ (Unsigned)	pc <= (reg[rs1] $>=_u$ reg[rs2]) ? label : pc + 4

Example 1: Arithmetic

C Code:

sum = (a – b) + (c + d)

Assume a, b, c, and d stored in x18, x19, x20, and x21, respectively

Answer:

```
sub x5, x18, x19
add x6, x20, x21
add x10, x5, x6
```

Example 2: If-Else

<u>C Code:</u>

```
if (x == 0) {
    y = 10
}
else {
    y = 20
}
```

Assume “x” stored in x5, “y” stored in x6

<u>Answer:</u>

```
        bne x5, x0, else
        addi x6, x0, 10
        jal x0, exit
else: addi x6, x0, 20
exit:
```

Example 3: While Loop

C Code:

```
x = 2
while (x >= 0) {
    y[x] = x + 1
    x = x - 1
}
```

Assume “x” stored in x5. Assume “y” is an array of integers (note: one int is 4 bytes) and starts at the address stored in x8.

Answer:

```
    addi x5, x0, 2
loop:
    blt x5, x0, exit
    addi x6, x5, 1
    slli x7, x5, 2
    add x7, x8, x7
    sw x6, 0(x7)
    addi x5, x5, -1
    beq x0, x0, loop
exit:
```

UCLA

Example 4: Assembly

<u>Assembly Code:</u>

```
li x5, 10
li x6, 5
add x7, x5, x6
sw x7, 3(x6)
lw x5, 3(x6)
addi x7, x5, -2
```

<u>Answer (register values at end of sequence):</u>

x5 = 15

x6 = 5

x7 = 13

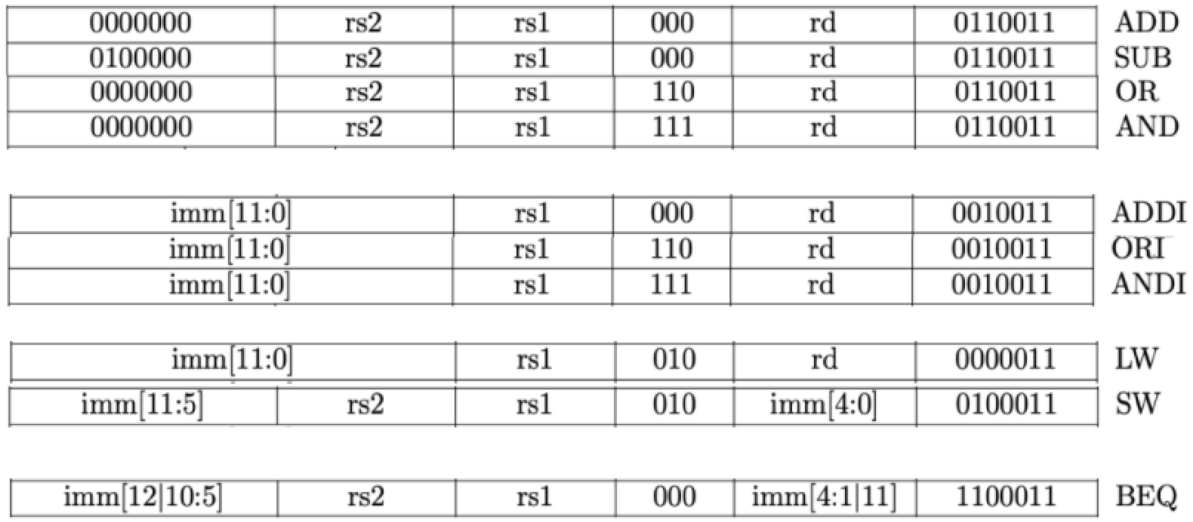
Example 5: Assembly to Machine Code

Assembly Code:

and x5, x10, x11

Machine Code:

0000000 01011 01010 111 00101 0110011



0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	110	rd	0010011	ORI
imm[11:0]	rs1	111	rd	0010011	ANDI

imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

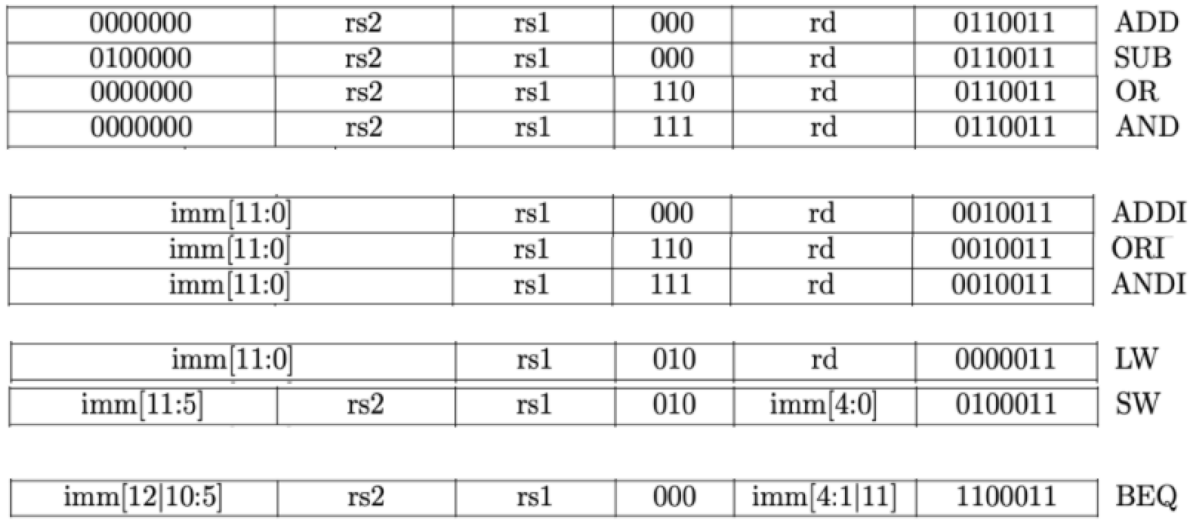
Example 6: Machine Code to Assembly

<u>Machine Code:</u>

00010000000 10010 000 01010 0010011

<u>Assembly Code:</u>

addi x10, x18, 128



0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ