Immediate Encodings

Constants / Immediates

- `lw` and `sw` use constants or *immediates*
- *immediate*ly available from instruction
- 12-bit two's complement number
- `addi`: add immediate
- **Is subtract immediate (`subi`) necessary?**

C Code

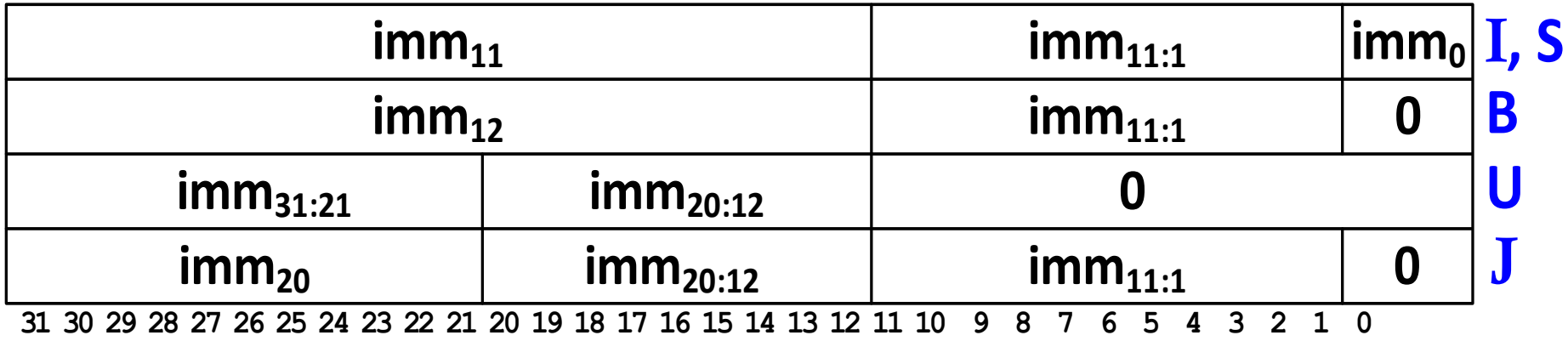
```
a = a + 4;
b = a - 12;
```

RISC-V assembly code

```
# s0 = a, s1 = b
addi s0, s0, 4
addi s1, s0, -12
```

Constants / Immediates

Immediate Bits



Immediate Encodings

Instruction Bits

funct7							4	3	2	1	0	rs1					funct3			rd								R
11	10	9	8	7	6	5	4	3	2	1	0	rs1					funct3			rd								I
11	10	9	8	7	6	5	rs2					rs1					funct3			4	3	2	1	0				S
12	10	9	8	7	6	5	rs2					rs1					funct3			4	3	2	1	11				B
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	rd								U
20	10	9	8	7	6	5	4	3	2	1	11	19	18	17	16	15	14	13	12	rd								J
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7				

- Immediate bits *mostly* occupy **consistent instruction bits**.
  - Simplifies hardware to build the microprocessor
- **Sign bit** of signed immediate is in **msb** of instruction.
- Recall that **`rs2`** of R-type can encode immediate shift amount.

Composition of 32-bit Immediates

Instruction Bits

funct7							4	3	2	1	0	rs1					funct3			rd						R
11	10	9	8	7	6	5	4	3	2	1	0	rs1					funct3			rd						I
11	10	9	8	7	6	5	rs2					rs1					funct3			4	3	2	1	0		S
12	10	9	8	7	6	5	rs2					rs1					funct3			4	3	2	1	11		B
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	rd						U
20	10	9	8	7	6	5	4	3	2	1	11	19	18	17	16	15	14	13	12	rd						J
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7		

instruction bit

31	31											31								31	30:25				24:21	20	I
31	31											31								31	30:25				11:8	7	S
31	31											31								7	30:25				11:8	0	B
31	30:20											19:12								0	0				0	0	U
31	31											19:12								20	30:25				24:21	0	J
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Immediate Bits

Chapter 6: Architecture

Reading

Machine Language & Addressing Operands

Instruction Fields & Formats

Instruction	op	funct3	Funct7	Type
`add`	0110011 (51)	000 (0)	0000000 (0)	R-Type
`sub`	0110011 (51)	000 (0)	0100000 (32)	R-Type
`and`	0110011 (51)	111 (7)	0000000 (0)	R-Type
`or`	0110011 (51)	110 (6)	0000000 (0)	R-Type
`addi`	0010011 (19)	000 (0)	-	I-Type
`beq`	1100011 (99)	000 (0)	-	B-Type
`bne`	1100011 (99)	001 (1)	-	B-Type
`lw`	0000011 (3)	010 (2)	-	I-Type
`sw`	0100011 (35)	010 (2)	-	S-Type
`jal`	1101111 (111)	-	-	J-Type
`jalr`	1100111 (103)	000 (0)	-	I-Type
`lui`	0110111 (55)	-	-	U-Type

See Appendix B for other instruction encodings

Interpreting Machine Code

- Write in binary
- Start with op: tells how to parse rest
- Extract fields
- op, funct3, and funct7 fields tell operation
- **Ex:** 0x41FE83B3 and 0xFDA58393

0x41FE83B3: 0100 0001 1111 1110 1000 0011 1011 0011

op = 51, funct3 = 0: add or sub (R-type)

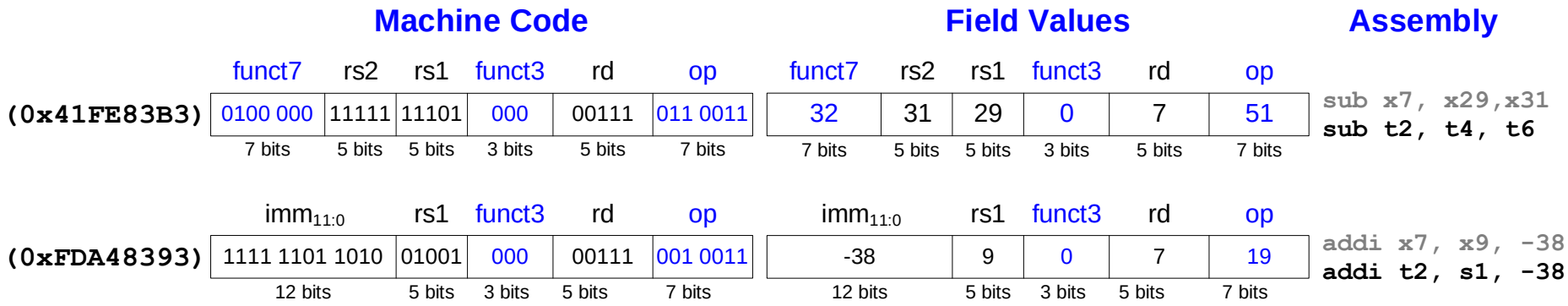
funct7 = 0100000: sub

0xFDA48393: 1111 1101 1010 0100 1000 0011 1001 0011

op = 19, funct3 = 0: addi (I-type)

Interpreting Machine Code

- Write in binary
- Start with op: tells how to parse rest
- Extract fields
- op, funct3, and funct7 fields tell operation
- **Ex:** 0x41FE83B3 and 0xFDA58393



Addressing Modes

How do we address the operands?

- Register Only
- Immediate
- Base Addressing
- PC-Relative

Addressing Modes

Register Only

- Operands found in registers
  - **Example:** `add s0, t2, t3`
  - **Example:** `sub t6, s1, 0`

Immediate

- 12-bit signed immediate used as an operand
  - **Example:** `addi s4, t5, -73`
  - **Example:** `ori  t3, t7, 0xFF`

Addressing Modes

Base Addressing

- Loads and Stores
- Address of operand is:

  `base address + immediate`

  - **Example:** `lw  s4, 72(zero)`
    - address = `0 + 72`

  - **Example:** `sw  t2, -25(t1)`
    - address = `t1 - 25`

Addressing Modes

PC-Relative Addressing: branches and `jal`

Example:

Address	Instruction
0x354	L1: addi s1, s1, 1
0x358	sub t0, t1, s7
...	...
0xEB0	bne s8, s9, L1

The label is (0xEB0-0x354) = 0xB5C (2908) instructions **before** `bne`

$imm_{12:0}$ = -2908 **1 0 1 0 0 1 0 1 0 0 1 0 1 0 0**

bit number **12 11 10 9 8 7 6 5 4 3 2 1 0**

Assembly	$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	Machine Code
bne s8, s9, L1	**1100 101**	24	25	1	**0010 0**	99	**(0xCB8C9263)**
(bne x24, x25, L1)	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
1100 101	11000	11001	001	**0010 0**	110 0011
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Chapter 6: Architecture

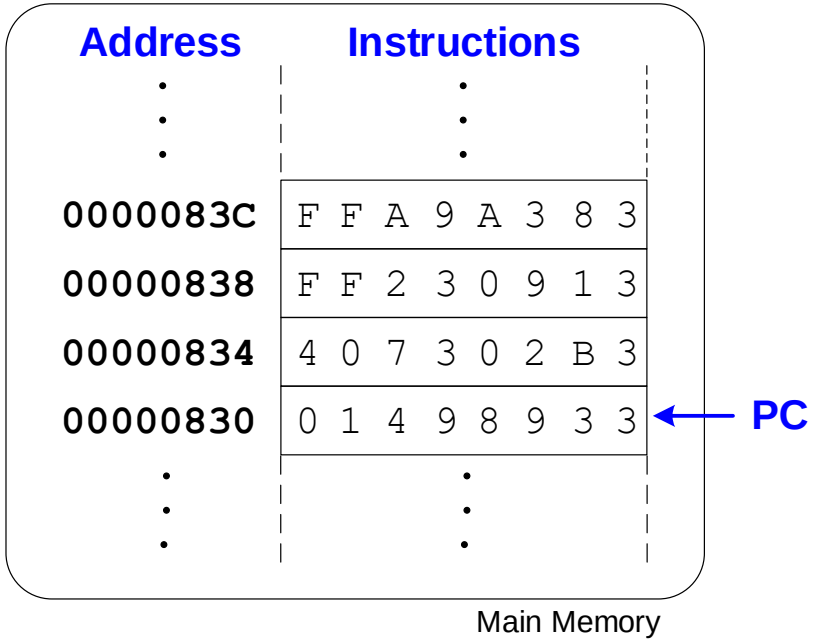
Compiling, Assembling, & Loading Programs

The Power of the Stored Program

- 32-bit instructions & data stored in memory
- Sequence of instructions: only difference between two applications
- To run a new program:
  - No rewiring required
  - Simply store new program in memory
- Program Execution:
  - Processor *fetches* (reads) instructions from memory in sequence
  - Processor performs the specified operation

The Stored Program

Assembly Code	Machine Code
`add  s2, s3, s4`	`0x01498933`
`sub  t0, t1, t2`	`0x407302B3`
`addi s2, t1, -14`	`0xFF230913`
`lw   t2, -6(s3)`	`0xFFA9A383`

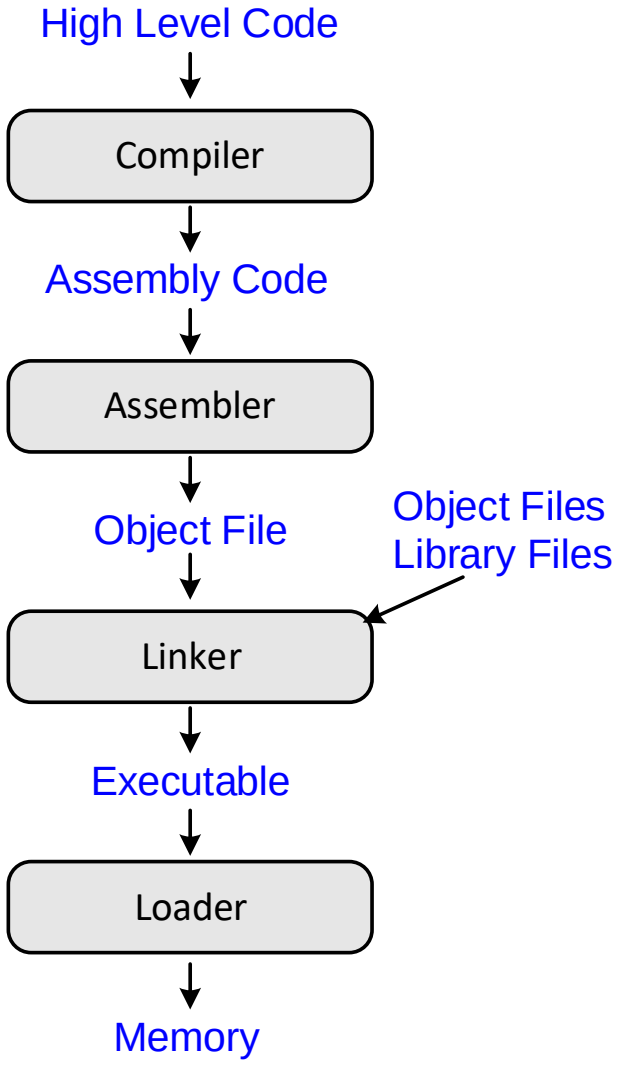


Program Counter (PC): keeps track of current instruction

Alan Turing, 1912 - 1954

- British mathematician and computer scientist
- Founder of theoretical computer science
- Invented the Turing machine: a mathematical model of computation
- Designed the Automatic Computing Engine, one of first stored program computers
- In 1952, was prosecuted for homosexual acts. Two years later, he died of cyanide poisoning.
- The Turing Award was named in his honor, which is the highest honor in computing.

How to Compile & Run a Program



Grace Hopper, 1906 - 1992

- Graduated from Yale University with a Ph.D. in mathematics
- Developed first compiler
- Helped develop the COBOL programming language
- Highly awarded naval officer
- Received World War II Victory Medal and National Defense Service Medal, among others

What is Stored in Memory?

- **Instructions** (also called *text*)
- **Data**
  - **Global/static:** allocated before program begins
  - **Dynamic:** allocated within program

- How **big** is memory?
  - At most 2^{32} = 4 gigabytes (4 GB)
  - From address 0x00000000 to 0xFFFFFFFF

Example RISC-V Memory Map