CSCI 210: Computer Architecture
Lecture 8: Computer Representation of RISC-V Instructions

Stephen Checkoway

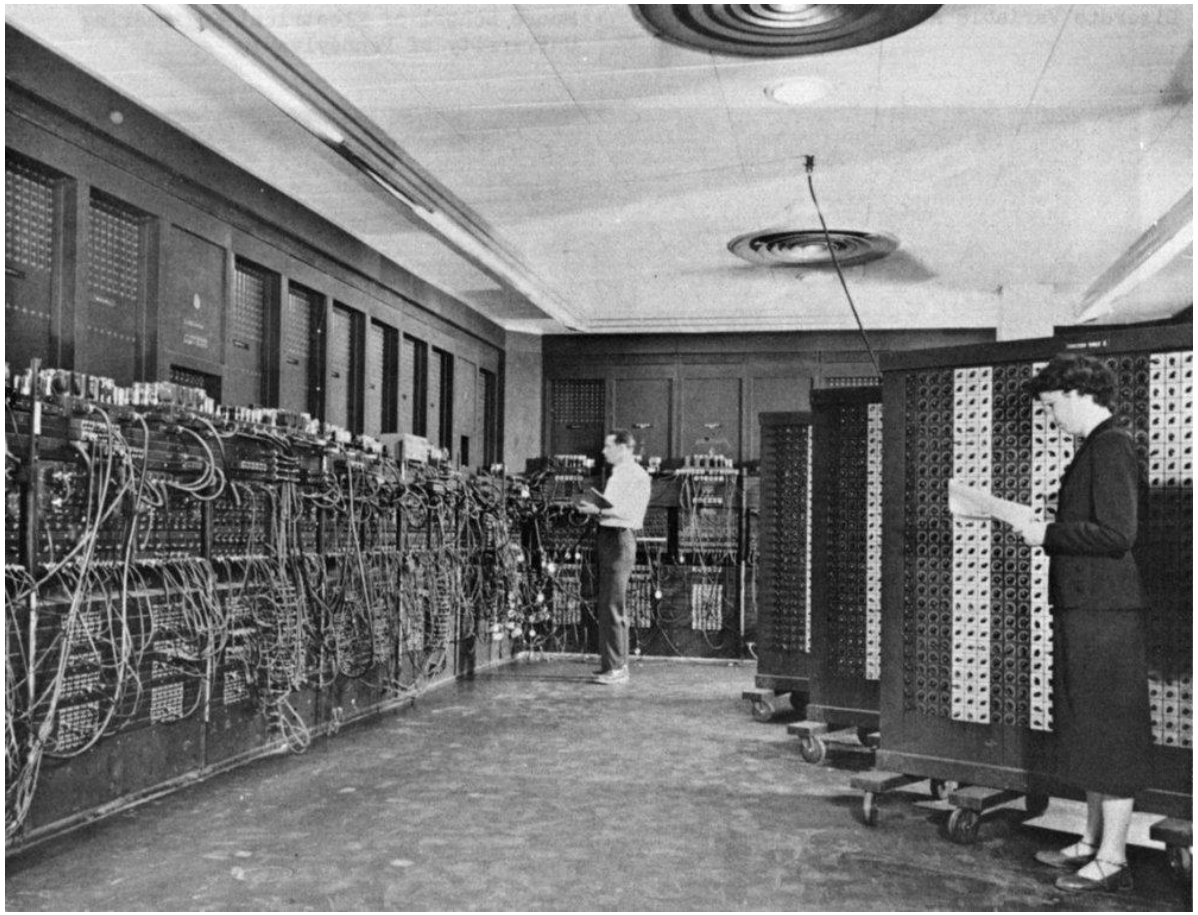
Oberlin College

Slides from Cynthia Taylor

Announcements

- Problem Set 2 due Friday

CS History: ENIAC



U.S. Army photo of ENIAC

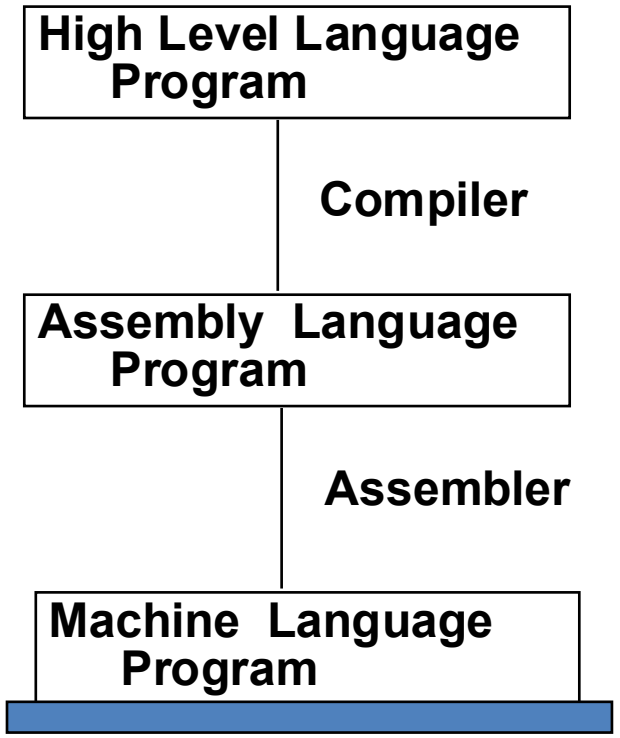
- Electronic Numerical Integrator And Computer
- First programmable, electronic, general-purpose computer
- Created by the US Army in 1945
- Designed to compute ballistic tables during WWII
- Originally didn’t have storage
- Decimal, not binary!

CS History: ENIAC

- Programmers were Kay McNulty, Jean Bartok, Betty Snyder, Marlyn Meltzer, Fran Bilas, and Ruth Lichterman.
- Selected from a group of 200 women employed hand calculating equations for the army
- Programmed by connecting components with cables and setting switches
- Kay McNulty developed the use of subroutines
- Betty Snyder and Jean Bartok went on to help develop the first commercial computers

U.S. Army photo

How to Speak Computer




```
temp = v[0];
v[0] = v[1];
v[1] = temp;
```

```
ld x14,0(x10)
ld x15,8(x10)
sd x14,8(x10)
sd x15,0(x10)
```

```
00000000000001010011011100000011
00000000100001010011011110000011
00000001110010100110100000100011
00000001111010100110000000100011
```

Machine Interpretation

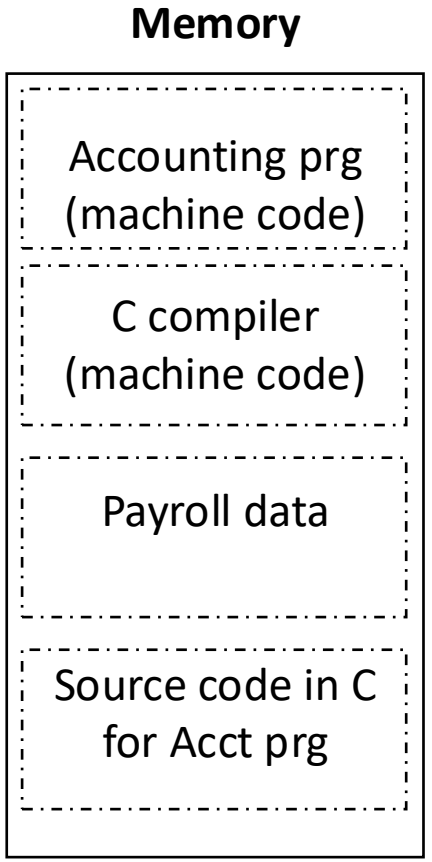
Two Key Principles of Machine Design

1. Instructions are represented as numbers and, as such, are indistinguishable from data

2. Programs are stored in alterable memory (that can be read or written to) just like data

Stored-program concept

- Programs can be shipped as files of binary numbers – binary compatibility
- Computers can inherit ready-made software provided they are compatible with an existing ISA and OS – leads industry to align around a small number of ISAs



What happens if someone writes new machine code in the memory where your program is stored, overwriting your program?

A. The program will crash.

B. The old instructions will run.

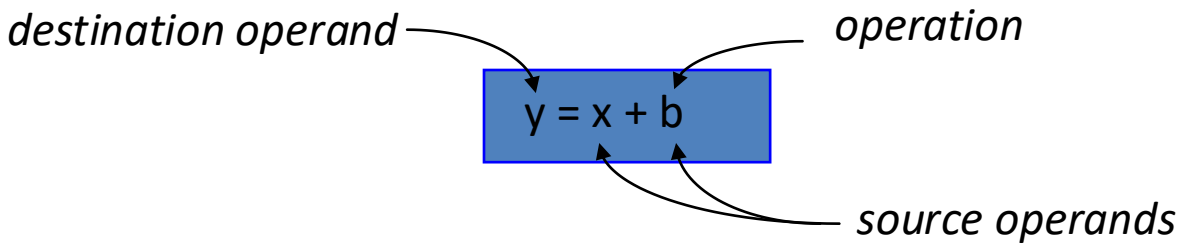
C. The new instructions will run.

D. None of the above

Recall: Instruction Set Architecture

- Definition of how to access the hardware from software
- Supported instructions, registers, etc . . .

Key ISA decisions



- operations
  - how many?
  - which ones
- operands
  - how many?
  - location
  - types
- instruction format
  - size
  - how many formats?

add r1, r2, r5

how does the computer know what
0001 0100 1101 1111
means?

RISC versus CISC (Historically)

- Complex Instruction Set Computing
  - Larger instruction set
  - More complicated instructions built into hardware
  - Variable number of clock cycles per instruction

- Reduced Instruction Set Computing
  - Small, highly optimized set of instructions
  - Memory accesses are specific instructions
  - One instruction per clock cycle (only the very first RISCs!)

A = A*B

RISC-V

```
ld    t0, 0(A)
ld    t1, 0(B)
mul   s1, t0, t1
sd    s1, 0(A)
```

CISC

```
mul   B, A
```

Which of these is faster?

RISC-V

```
ld    t0, 0(A)
ld    t1, 0(B)
mul   s1, t0, t1
sd    s1, 0(A)
```

CISC

```
mul   B, A
```

RISC vs CISC

RISC

- More work for compiler/assembly programmer
- More RAM used to store instructions
- Less complex hardware

CISC

- Less work for compiler/assembly programmer
- Fewer instructions to store
- More complex hardware

So . . . Which System “Won”?

- Most processors are RISC
- BUT the x86 (Intel) is CISC
- x86 breaks down CISC assembly into multiple, RISC-like, machine language instructions
- Distinction between RISC and CISC is less clear
  - Some RISC instruction sets have more instructions than some CISC sets

The computer figures out what format an instruction is from

A. Codes embedded in the instruction itself.

B. A special register that is loaded with the instruction.

C. It tries each format and sees which one forms a valid instruction.

D. None of the above

Instruction Formats What does each bit mean?

- Having many different instruction formats…
  - complicates decoding
  - uses more instruction bits (to specify the format)

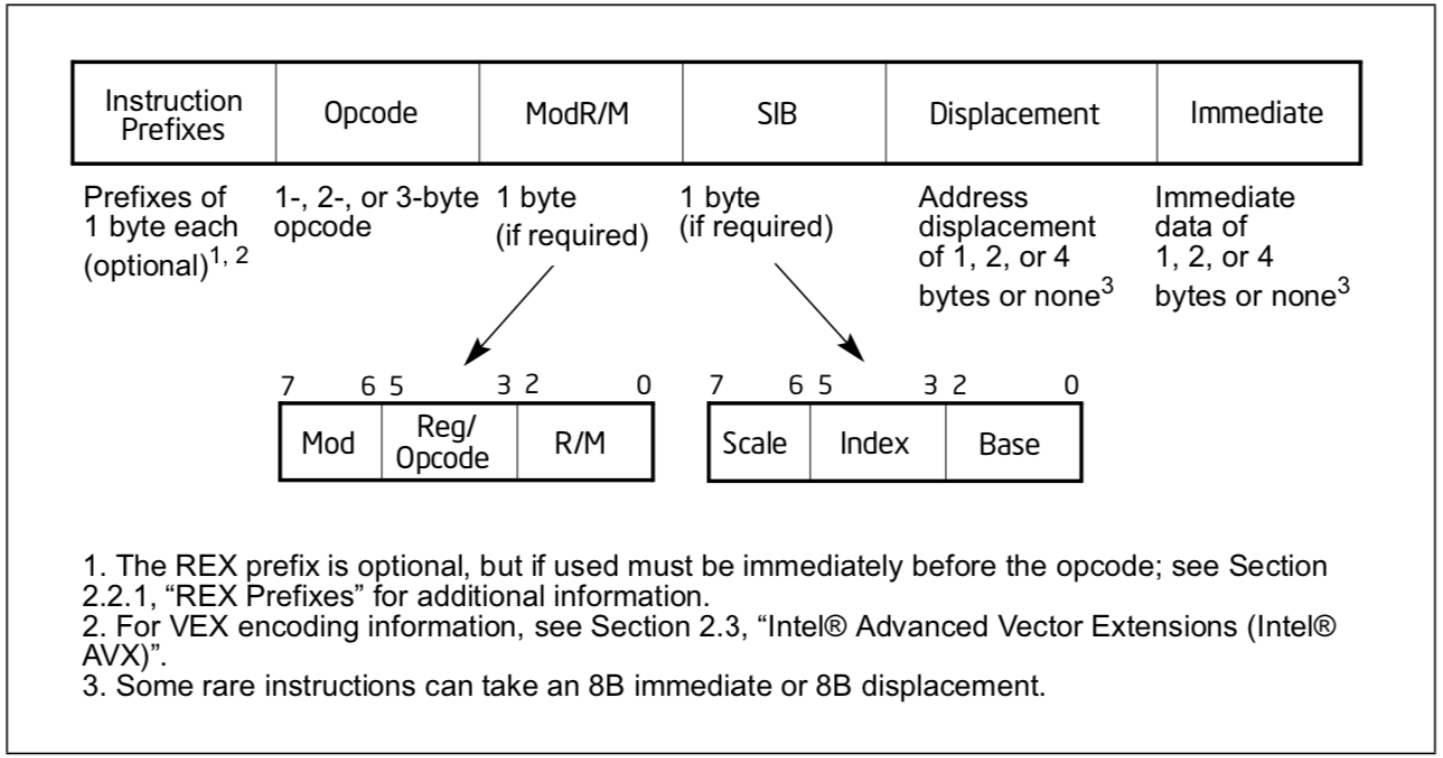


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

x86-64 example

```
Encoding          Instruction
01 d8             add eax, ebx
48 01 d8          add rax, rbx
48 03 03          add rax, qword ptr [rbx]
48 03 04 8b       add rax, qword ptr [rbx + 4*rcx]
48 03 44 8b 18    add rax, qword ptr [rbx + 4*rcx + 0x18]
```

REX prefix specifying 64-bit registers

Opcode specifying the instruction

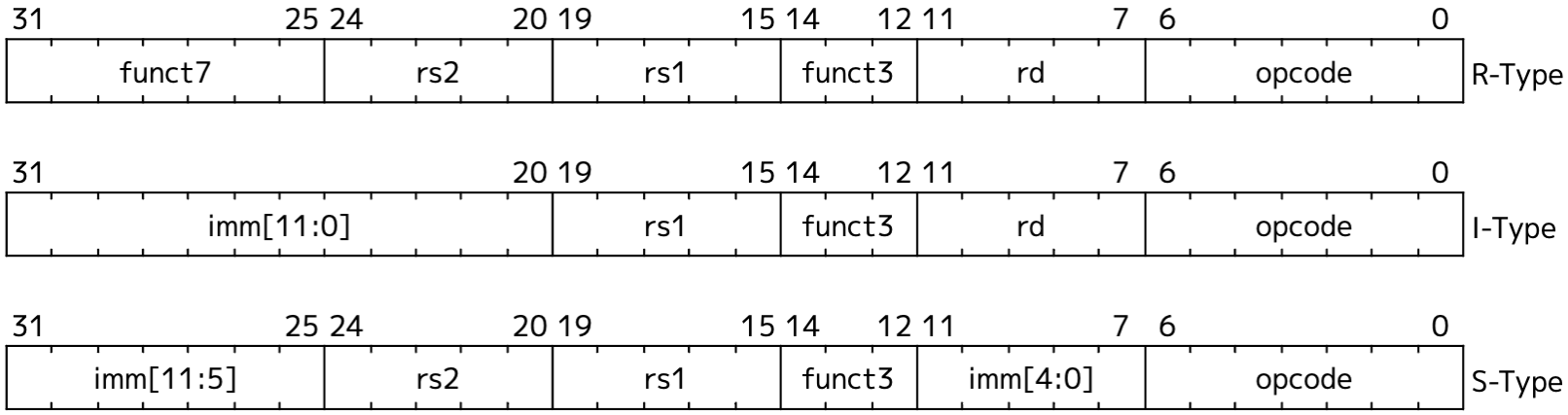
ModR/M specifying the operands (including reg vs. mem)

SIB specifying the scale, index register, and base register

Displacement (offset)

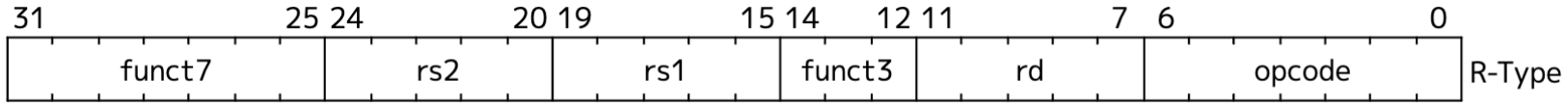
Representing Instructions

- RISC-V instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity leads to simplicity of hardware
- We'll look at 3 of the 6 formats today



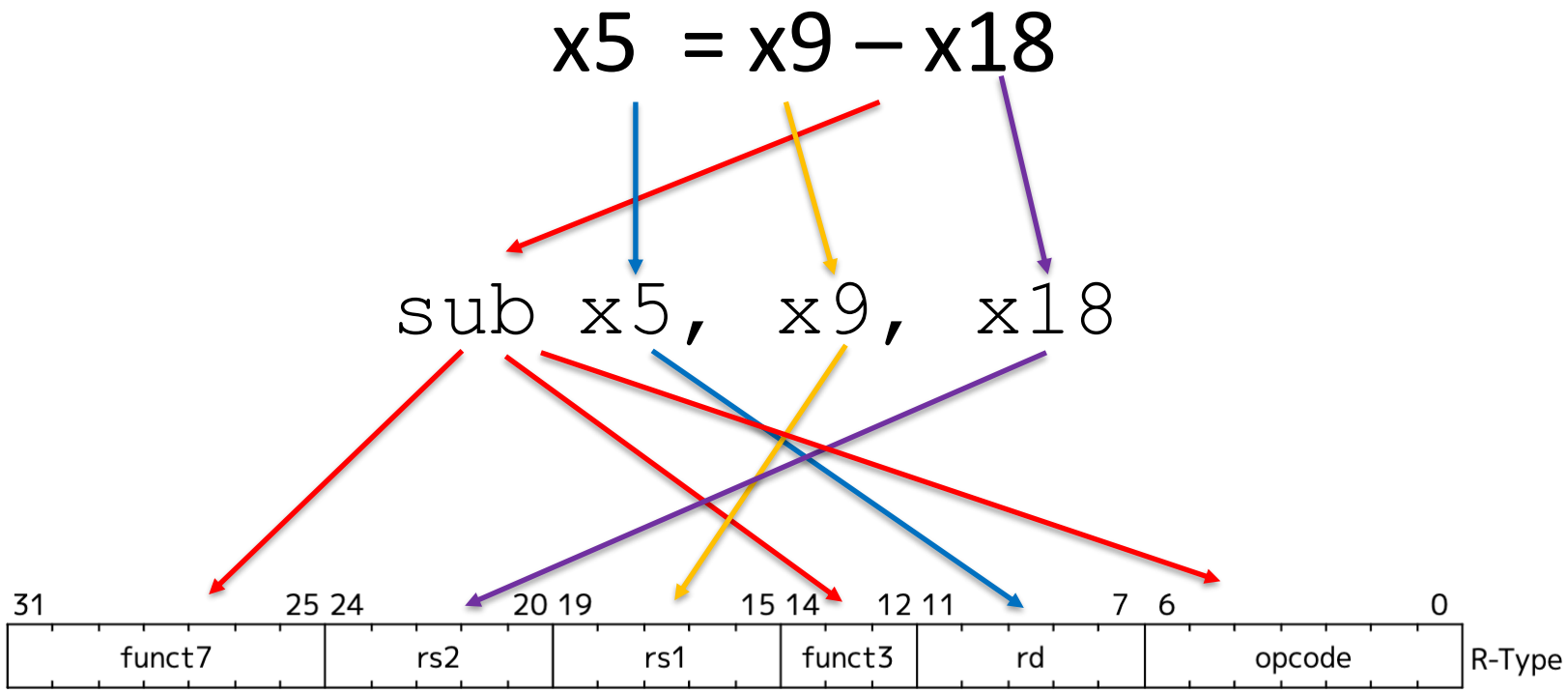
RISC-V Instruction Fields for R-type

- RISC-V instructions fields are given names to make them easier to refer to



opcode	7-bits	opcode that specifies the operation (along with funct3 and funct7)
funct3	3-bits	additional bits used to specify the operation
funct7	7-bits	additional bits used to specify the operation
rd	5-bits	register number for the destination operand
rs1	5-bits	register number for the first source operand
rs2	5-bits	register number for the second source operand

RISC-V Arithmetic Instructions Format



For sub, the opcode, funct3, and funct7 are 0x33, 0x0, and 0x20

rd = 5 = 0x05

rs1 = 9 = 0x09

rs2 = 18 = 0x12

In binary: 0100000_10010_01001_000_00101_0110011

The RISC-V Reference Data Card

- From the back of the textbook, also on the course website
- Front side gives a description of each instruction
- Back side gives register and instruction encoding

RISC-V ①

Reference Data

RV64I BASE INTEGER INSTRUCTIONS, in alphabetical order

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	NOTE
add,addw	R	ADD (Word)	R[rd] = R[rs1] + R[rs2]	1)
addi,addiw	I	ADD Immediate (Word)	R[rd] = R[rs1] + imm	1)
and	R	AND	R[rd] = R[rs1] & R[rs2]	
andi	I	AND Immediate	R[rd] = R[rs1] & imm	

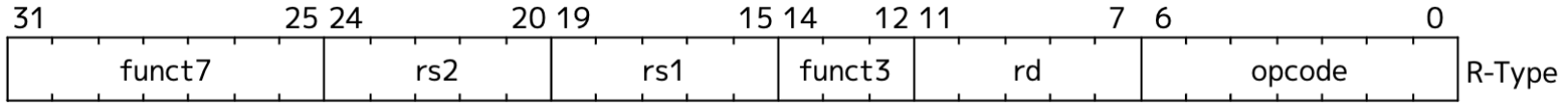
REGISTER NAME, USE, CALLING CONVENTION

REGISTER	NAME	USE
x0	zero	The constant value 0
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5-x7	t0-t2	Temporaries
x8	s0/fp	Saved register/Frame pointer
x9	s1	Saved register
x10-x11	a0-a1	Function arguments/Return values
x12-x17	a2-a7	Function arguments
x18-x27	s2-s11	Saved registers
x28-x31	t3-t6	Temporaries
f0-f7	ft0-ft7	FP Temporaries
f8-f9	fs0-fs1	FP Saved registers

OPCODES IN NUMERICAL ORDER BY OPCODE

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIMAL
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2
ld	I	0000011	011		03/3

R-format Example




```
add a0, a1, a2
```

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)
add, addw	R	ADD (Word)	R[rd] = R[rs1] + R[rs2]
addi, addiw	I	ADD Immediate (Word)	R[rd] = R[rs1] + imm

REGISTER NAME, USE, CALLING CONVENTION

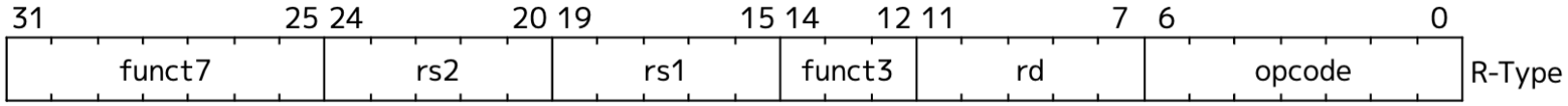
REGISTER	NAME	USE
x0	zero	The constant value 0
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5-x7	t0-t2	Temporaries
x8	s0/fp	Saved register/Frame pointer
x9	s1	Saved register
x10-x11	a0-a1	Function arguments/Return values
x12-x17	a2-a7	Function arguments
x18-x27	s2-s11	Saved registers
x28-x31	t3-t6	Temporaries
f0-f7	ft0-ft7	FP Temporaries
f8-f9	fs0-fs1	FP Saved registers

1. Look up the values for each field
   - opcode, funct3, funct7 = 0x33, 0x0, 0x00
   - rd = 10
   - rs1 = 11
   - rs2 = 12
2. Convert each field to binary
3. Concatenate all of the bits in the appropriate order
   `0000000_01100_01011_000_01010_0110011`
4. (Optional) Group into groups of 4 bits and convert to hex
   `0000_0000_1100_0101_1000_0101_0011_0011 = 0x00C58533`

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIMAL
add	R	0110011	000	0000000	33/0/00

Convert this RISC-V machine instruction to assembly:

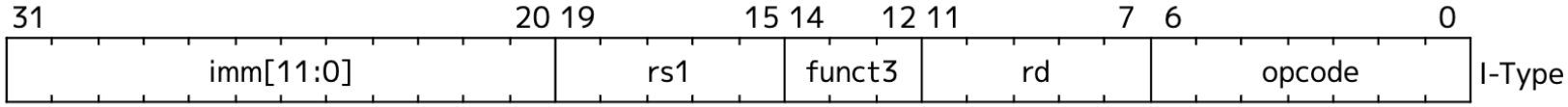
```
0000000_01010_01000_111_00101_0110011
```



Selection	Instruction
A	`and t0, s0, a0`
B	`slt a0, s0, t0`
C	`add s0, t0, a0`
D	`sub t0, s0, a0`
E	None of the above

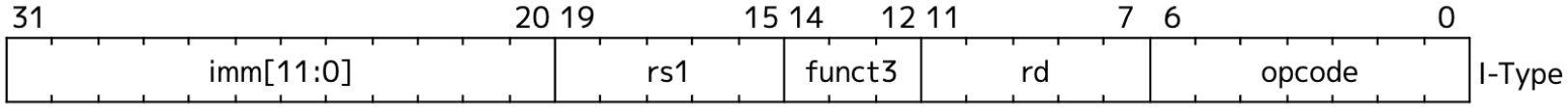
RISC-V Instruction Fields for I-format

- I-format is used for arithmetic/logical instructions with an immediate operand as well as load instructions



opcode	7-bits	opcode that specifies the operation (along with funct3)
funct3	3-bits	additional bits used to specify the operation
rd	5-bits	register number for the destination operand
rs1	5-bits	register number for the first source operand
imm	12-bits	signed immediate value used for the second source operand

Machine Language – I Format




```
addi   x5, x0, -1024
```

opcode, funct3 = 0x13, 0x0

rd = 5, rs1 = 0, imm = 110000000000 (binary)

110000000000_00000_000_00101_0010011

```
ld     t0, 8(sp)
```

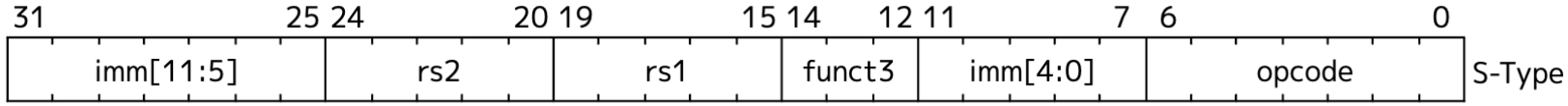
opcode, funct3 = 0x03, 0x3

rd = 5, rs1 = 2, imm = 000000001000 (binary)

000000001000_00010_011_00101_0000011

RISC-V Instruction Fields for S-format

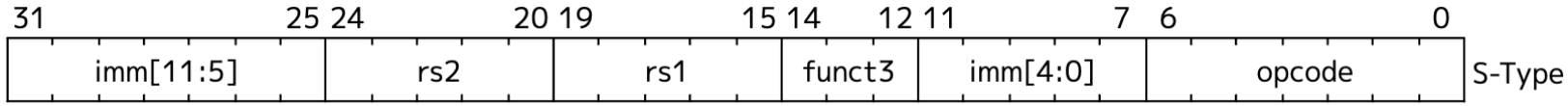
- S-format is used for store instructions `sb`, `sh`, `sw`, and `sd`



opcode	7-bits	opcode (always 0x23) that specifies the store operation
funct3	3-bits	additional bits used to specify the size of the store (byte, half, etc.)
rs1	5-bits	register number for the base address
rs2	5-bits	register number for the register whose value is stored to memory
imm	12-bits	signed immediate value used for the offset

Note that the 12-bit immediate is split into two fields imm[11:5] which holds the most significant 7 bits of the offset and imm[4:0] which holds the least significant 5 bits

Machine Language – S Format




```
sd   t3, 1024(s0)
```

opcode, funct3 = 0x23, 0x3

rs1 (s0) = 0x08, rs2 (t3) =0x1C

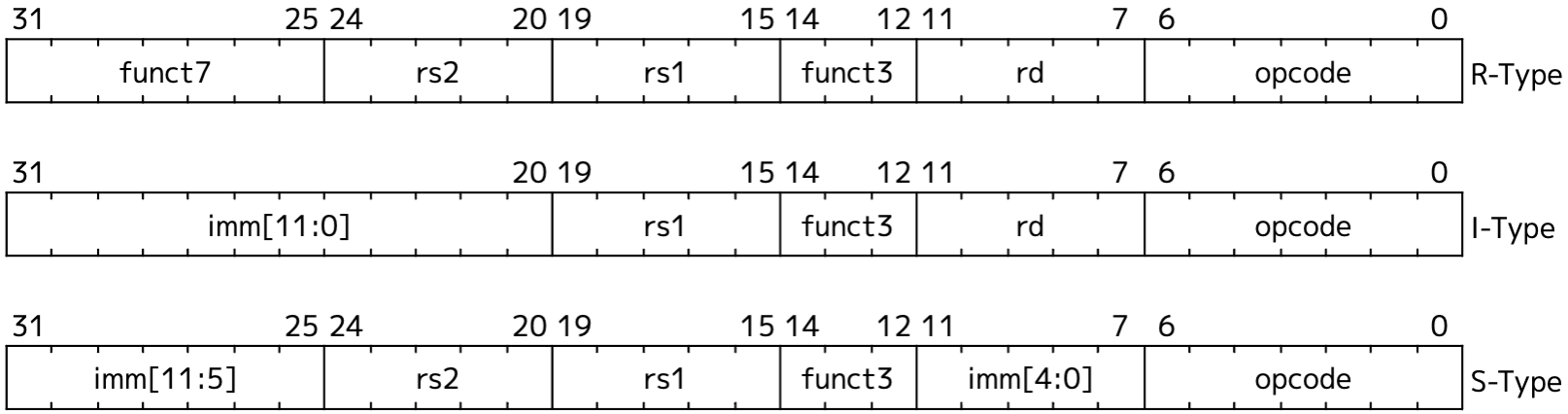
imm = 0100000_00000 (binary, split into 7-5 bits)

0100000_011100_001000_011_00000_0100011

Take aways

- Every instruction has an opcode in the least significant 7 bits of the instruction word
- The opcode specifies which of the 6 instruction formats (of which we've seen 3, we'll look at the others later) the instruction uses as well as the operation to perform
- R-, I-, and S-format instructions have an additional 3-bit field, `funct3`, used to further specify the instruction (often used to indicate the size of the operation, e.g., `lw` vs. `ld` or `sw` vs. `sd`
- R-format instructions have an additional 7-bit field, `funct7`, used to further specify the instruction

Group discussion questions



Compare the 3 instruction formats

- What fields do they have in common?
- Why do you think store instructions have their own format that splits the immediate operand into two fields rather than reuse the I-format used by load instructions which doesn't split it?
- Why do you think there's a separate `funct3` field rather than just making the opcode 10 bits and why isn't `funct3` next to `opcode`?

In groups, convert 0xFFC12823 into binary and work out what RISC-V instruction it is

Start by figuring out the opcode, then look up what format instruction it is, then decode all of the fields

Select any answer when you have it

Questions about Machine Instructions?

Reading

- Next lecture: Bitwise Operations
  - Section 2.7

- Problem Set 2 due Friday