National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Lecture 2: RISC-V Instruction Set, Part 1

CS10014 Computer Organization

Department of Computer Science
Tsung Tai Yeh
Thursday: 1:20 pm– 3:10 pm
Classroom: EC-022

Acknowledgements and Disclaimer

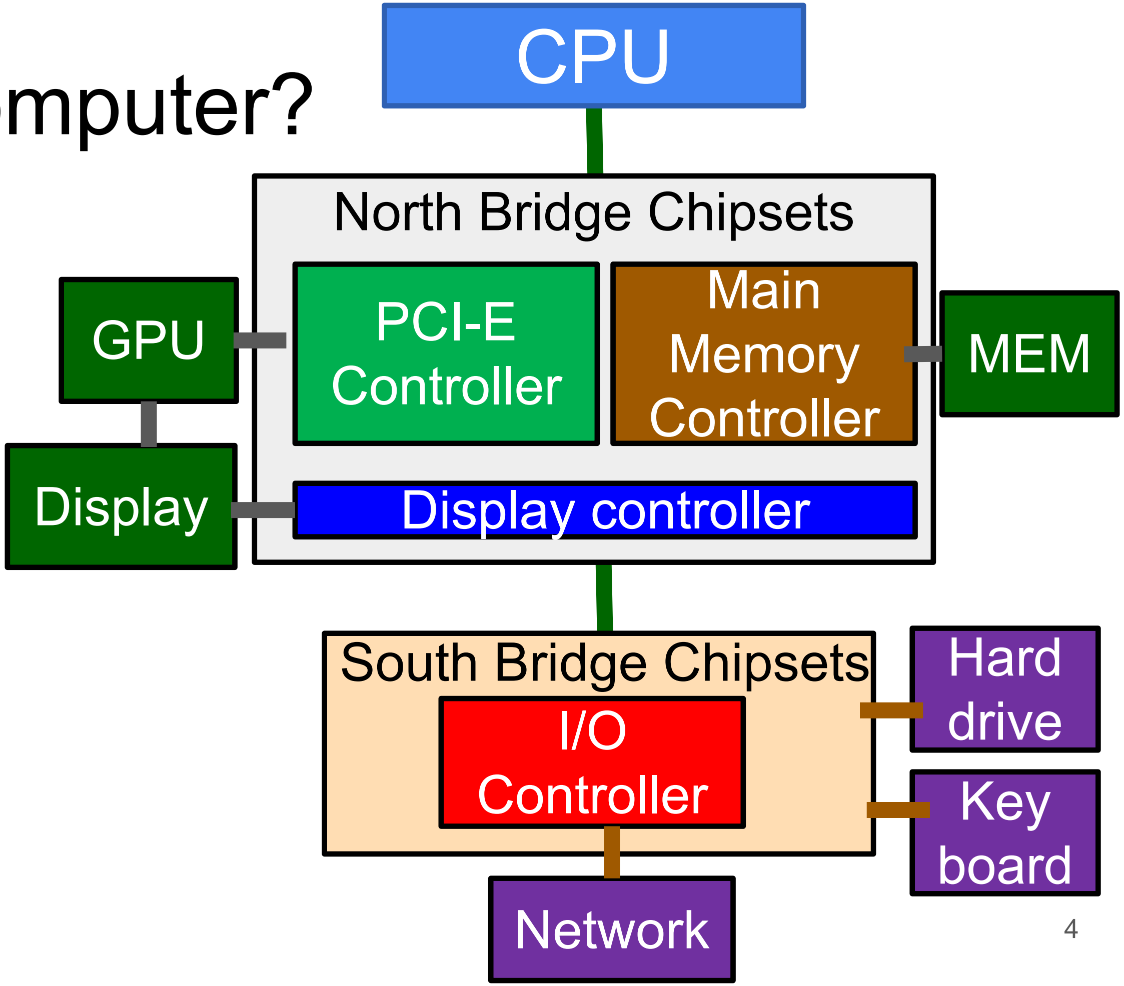
- Slides were developed in the reference with
    - CS 61C at UC Berkeley
        - https://inst.eecs.berkeley.edu/~cs61c/sp23/
    - CS 252 at UC Berkeley
        - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
    - CSCE 513 at University of South Carolina
        - https://passlab.github.io/CSCE513/

Outline

- RISC vs. CISC
- RISC-V Registers
- Basic Arithmetic Instructions
- Immediates
- Data Transfer Instructions

What is inside a computer?

- A Computer
  - Processor
  - Main memory
  - I/O devices
- North Bridge Chipsets
  - High-speed channels
  - PCI-E/AGP
  - Memory controller hub
- South Bridge Chipsets
  - I/O channel
  - I/O controller hub



Processor Architecture

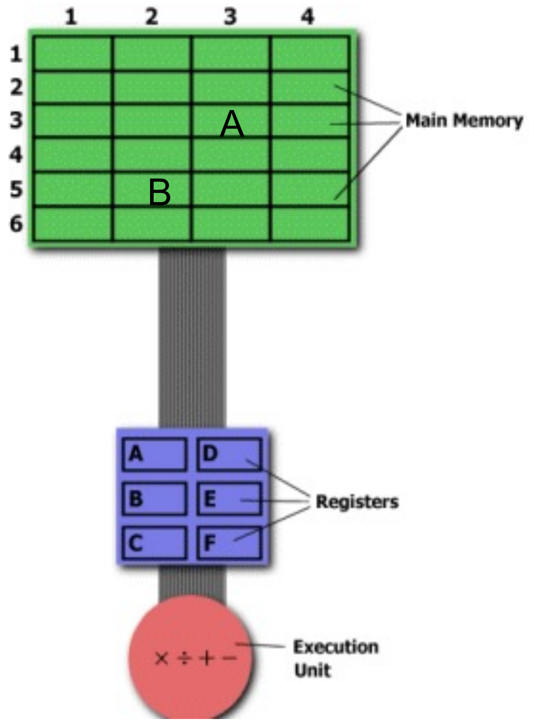
- Instruction Set Architecture (ISA)
  - Determine the operations (instructions) in a processor
- Micro-architecture
  - Refers to the internal organization of a specific processor

Different Instruction Set Architecture

- ARM
  - Family of ISAs developed by ARM
  - Used in embedded systems (mobile and low power) and desktop (Apple M1/2)
- x86
  - Family of ISAs developed by Intel (and AMD)
  - Used in general-purpose computing systems (desktop and servers)
- RISC-V
  - Open standard ISAs developed by UC-Berkeley
  - Mostly used in embedded systems

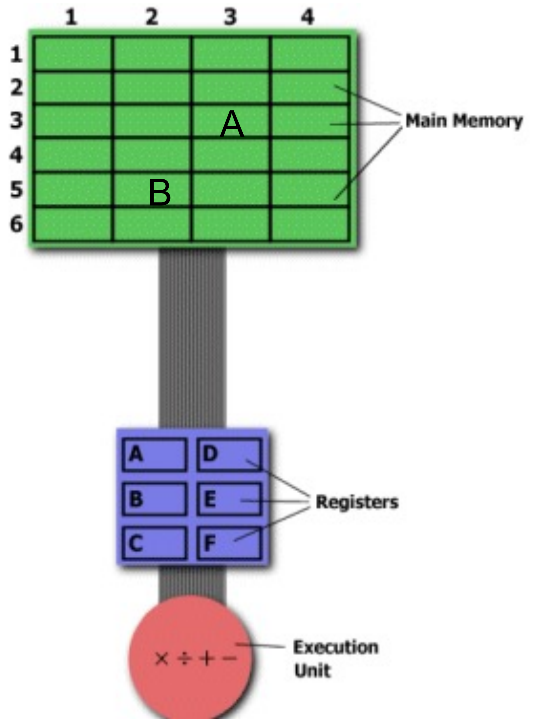
RISC vs. CISC

- CISC (Complex Instruction Set Computers)
  - Complete a task in as few lines of assembly as possible
  - CISC processor should execute a series of operations
  - E.g. MULT 2:3, 5:2
    - This MULT instruction loads two values into separate registers
    - Multiplies the operands in the execution unit
    - Stores the product in the appropriate register
    - The entire task of multiplying two number can be completed with on instruction



RISC vs. CISC

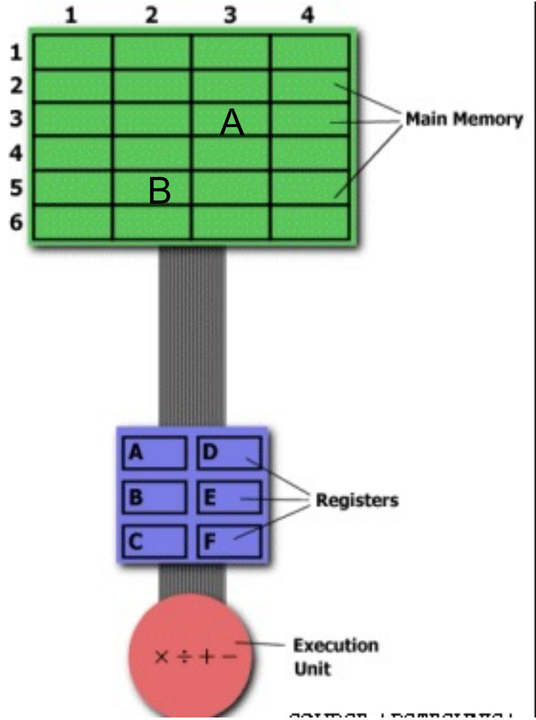
- CISC (Complex Instruction Set Computers)
    - The compiler has to do very little work to translate a high-level language statement into assembly
    - The length of the code is short
    - Very little RAM is required to store instructions
    - The emphasis is put on building complex instructions directly into the hardware



https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

RISC vs. CISC

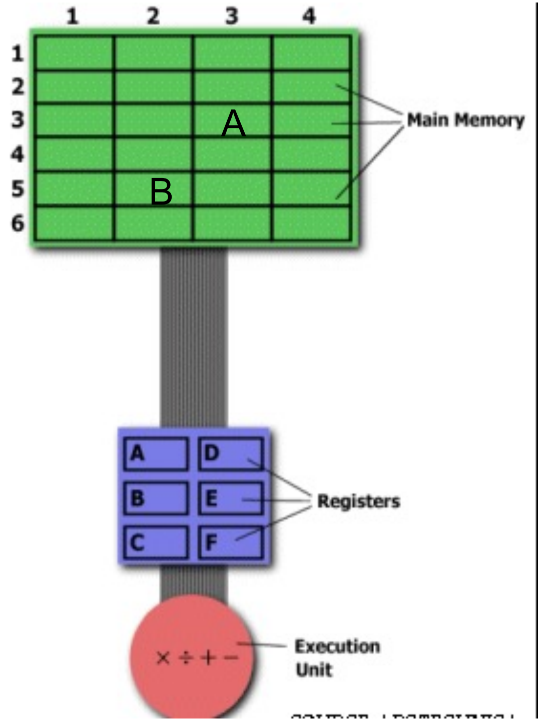
- RISC (Reduced Instruction Set Computers)
  - Use simple instructions that can be executed within one clock cycle
  - The “MULT” is divided into three separate instructions
    - LOAD: moves data from the memory bank to a register
    - PROD: finds the product of two operands located within the registers
    - STORE: moves data from a register to the memory banks



```
LOAD    A,   2:3
LOAD    B,   5:2
PROD    A,   B
STORE  2:3, A
```

RISC vs. CISC

- RISC (Reduced Instruction Set Computers)
  - Require fewer transistors of hardware space than complex instructions (why?)
  - Separate the “LOAD” and “STORE” instructions reduces the amount of work that the computer must perform (why?)
    - “MULT” automatically erases the registers
    - If one of the operands needs to be used for another computation, the processor must re-load the data from the memory into a register



```
LOAD    A,   2:3
LOAD    B,   5:2
PROD    A,   B
STORE   2:3, A
```

Summary -- RISC vs. CISC

- CISC
  - One instruction will take complicated work
  - Let the hardware do the complicated operations
  - Super-complicated (Slow?) hardware
- RISC
  - A simpler (and small) instruction set makes it easier to build fast hardware
  - Let the software do the complicated operations by composing simpler ones

What is RISC-V?

- RISC (Reduced Instruction Set Computers)
  - Fifth generation of RISC design from UC Berkeley
  - A license-free, royalty-free RISC ISA specification
    - Licensing cost for the ISA can be in the noises
  - Appropriate for all levels of computer systems, from micro-controllers to supercomputers
    - 32-bit, 64-bit, and 128-bit variants
    - We use 32-bit in the class
  - Standard maintained by non-profit RISC-V Foundation

The RISC-V Instruction Set

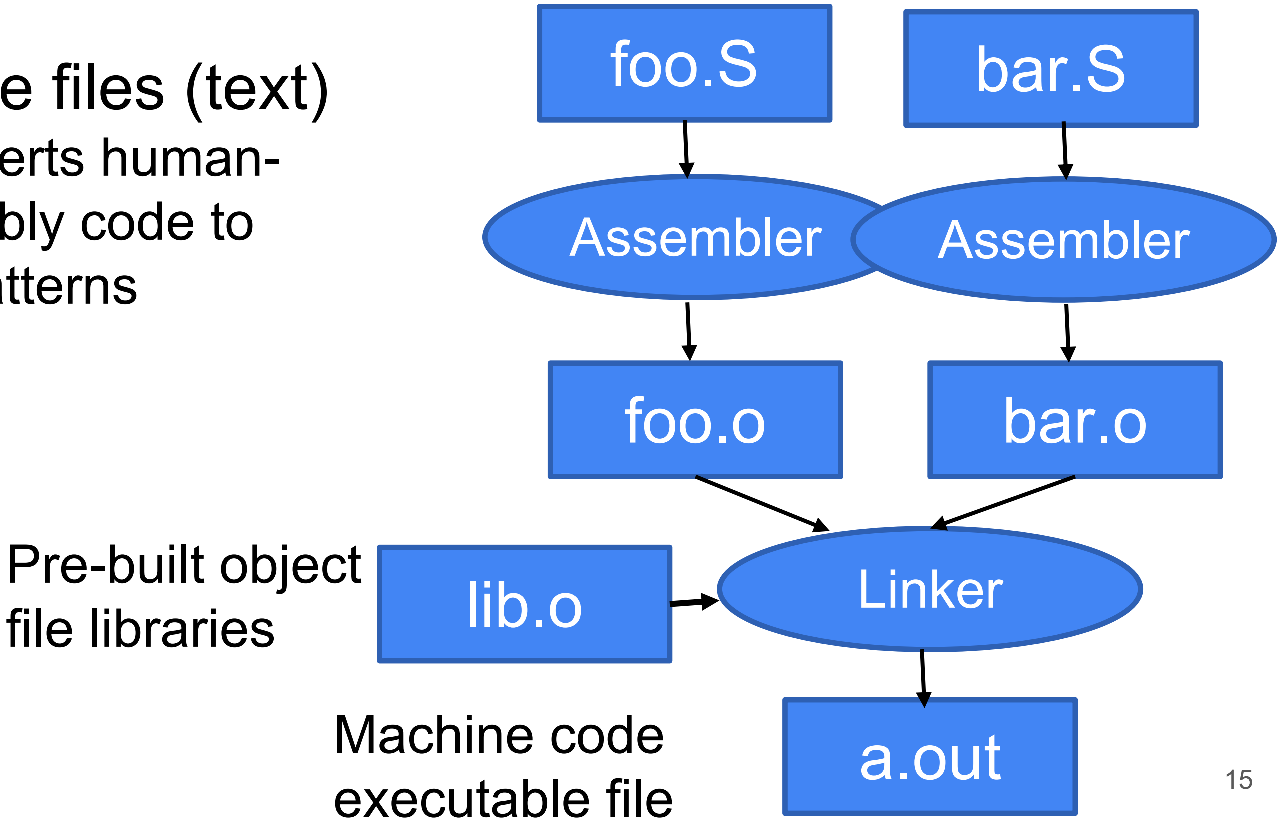
- Fixed size of 32 bits
- 32 registers
- Load-store architecture
  - Arithmetic and logic instructions working on registers only
  - Specific instructions to move data from and to memory
- RISC ISA
  - Few (49) instructions in the base ISA
  - Regular instruction formats
  - Allows for very small hardware implementations

The RISC-V Instruction Set

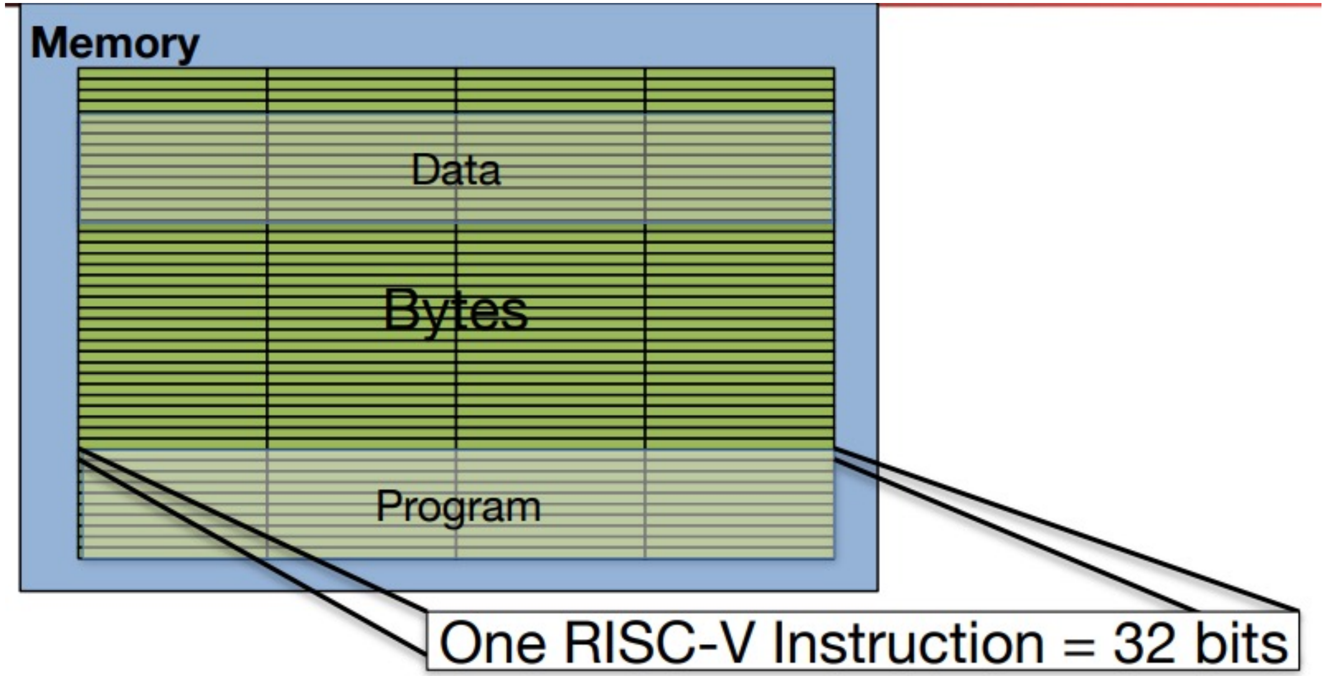
- Standardized extensions of base ISA
  - Multiplication and division
  - Floating point
  - Bit manipulation
  - Vector computations
  - …

Assembler to Machine Code

- Assembler source files (text)
  - Assembler converts human-readable assembly code to instruction bit patterns

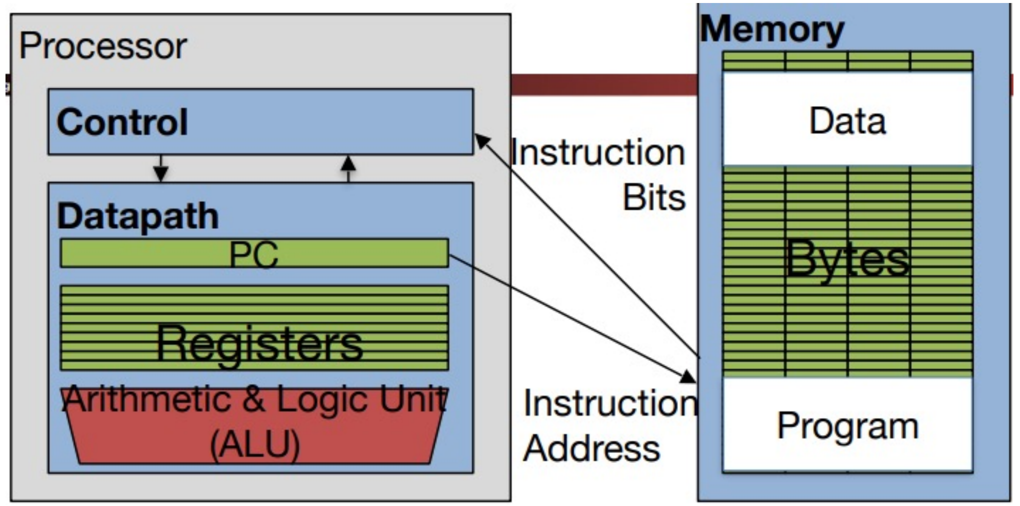


How Program is Stored



Program Execution

- PC (program counter)
  - A special internal register inside the processor holding the byte address of the next instruction to be executed
  - Instruction is fetched from memory
  - Control unit executes instruction using datapath and memory the system and updates PC
  - Add +4 bytes to PC
    - Move to next sequential inst.



Outline

- RISC vs. CISC
- RISC-V Registers
- Basic Arithmetic Instructions
- Immediates
- Data Transfer Instructions

Variables in Hardware

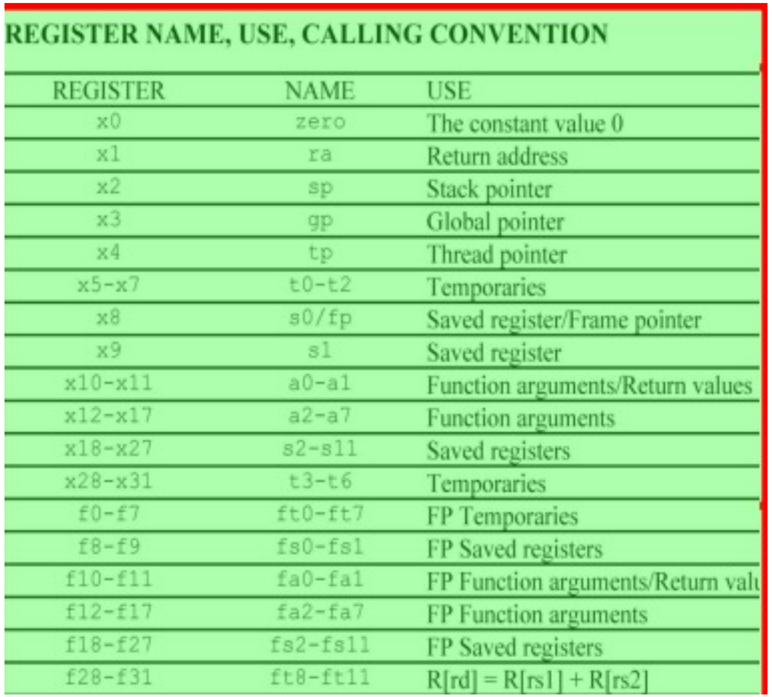
- Instructions must be directly implemented in hardware
- What about variables?
  - Live in memory (DRAM)
  - Memory is slow compared to the processor
  - Predetermined small number of fast **registers** in processor to hold variables

Registers vs. Memory

- What if more variables than registers?
  - Keep most frequently used in registers and move the rest to memory (called spilling to memory)
- Why not all variables in memory?
  - Smaller is faster: registers 100-500 times faster
  - Registers more versatile
    - In 1 arithmetic instruction: read 2 operands, perform 1 operation, and 1 write
    - In 1 data transfer instruction: 1 memory read/write, no operation

How Many Registers?

- Tradeoff between speed and availability
  - More registers -> can house more variables simultaneously
  - Why 32? Smaller is faster, but too small is bad
    - Need to specify 3 registers in operations
- RISC-V has 32 registers (x0 – x31)
  - Each register is 32 bits wide and holds a word



REGISTER NAME, USE, CALLING CONVENTION

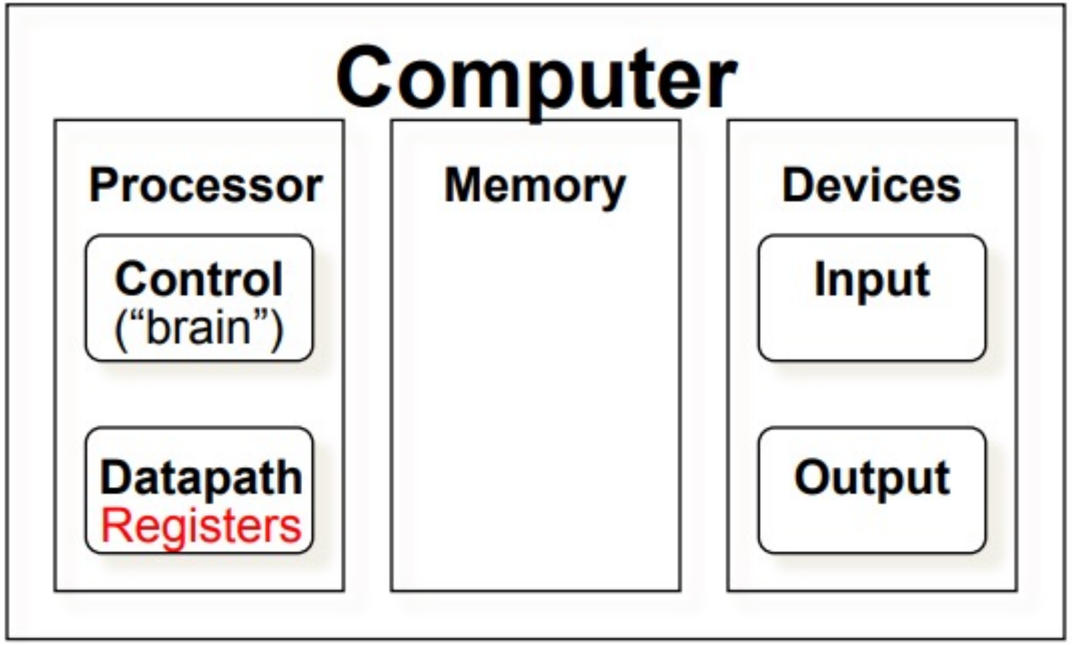
REGISTER	NAME	USE
x0	zero	The constant value 0
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5-x7	t0-t2	Temporaries
x8	s0/fp	Saved register/Frame pointer
x9	s1	Saved register
x10-x11	a0-a1	Function arguments/Return values
x12-x17	a2-a7	Function arguments
x18-x27	s2-s11	Saved registers
x28-x31	t3-t6	Temporaries
f0-f7	ft0-ft7	FP Temporaries
f8-f9	fs0-fs1	FP Saved registers
f10-f11	fa0-fa1	FP Function arguments/Return valu
f12-f17	fa2-fa7	FP Function arguments
f18-f27	fs2-fs11	FP Saved registers
f28-f31	ft8-ft11	R[rd] = R[rs1] + R[rs2]

RISC-V Registers

- Register denoted by ‘x’ can be referenced by number (x0-x31) or name
  - Register that hold programmer variables
    - s0 – s1 <-> x8 – x9
    - s2 – s11 <-> x18-x27
  - Registers that hold temporary variables
    - t0 – t2 <-> x6 – x7
    - t3 – t6 <-> x28-x31
- Register have no type
  - The operation being performed determines how register contents are treated

Registers live inside the processor

- Registers are part of the datapath



Registers -- Summary

- In high-level languages, the number of variables is limited only by available memory
- ISAs have a fixed, small number of operands call registers
    - Special locations built directly into hardware
    - **Benefit:**
        - Registers are EXTREMELY FAST
    - **Drawback:**
        - Operations can only be performed on these predetermined number of registers

Outline

- RISC vs. CISC
- RISC-V Registers
- Basic Arithmetic Instructions
- Immediates
- Data Transfer Instructions

RISC-V Instruction (1/2)

- Instruction Syntax is rigid

  op dst, src1, src2

  - 1 operator, 3 operands
    - op = operation name (“operator”) such add, sub …
    - dst = register getting the result (“destination”)
    - src1 = first register for operation (“source 1”)
    - src2 = second register for operation (“source 2”)
- Keep hardware simple via regularity

RISC-V Instruction (2/2)

- One operation per instruction
  - At most one instruction per line
- Assembly instructions are related to C
  - Operations (=, +, -, *, /, &, |, etc.)
  - A single line of C may break up into several lines of RISC-V

RISC-V Instructions Example

- Integer Addition (add)
  - Example: **add x1, x2, x3** (in RISC-V)
  - Equivalent to a = b + c (in C)
    where C variables ⇔ RISC-V registers are:
    a ⇔ x1, b ⇔ x2, c⇔ x3
- Integer Subtraction (sub)
  - Example: **sub x3, x4, x5** (in RISC-V)
  - Equivalent to d = e – f
    where C variables ⇔RISC-V registers are:
    d ⇔ x3, e⇔ x4, f ⇔ x5

RISC-V Instructions Example

- Suppose a->s0, b->s1, c->s2, d->s3, and e->s4
  - C-code
    - a = (b + c) - (d + e)
  - RISC-V codes

```
add   t1,   s3,    s4
add   t2,   s1,    s2
sub   s0,   t2,    t1
```

Utilize temporary registers

Ordering of instructions matters (must follow the order of operations)

Comments in RISC-V

- Comments in RISC-V follow the hash mark (#) until the end of the line
  - Improve readability and helps you keep track of variables/registers
  - C-code
    - a = (b + c) - (d + e)
  - RISC-V codes

```
add   t1,   s3,    s4   # temp1 = d + e
add   t2,   s1,    s2   #temp2 = b + c
sub   s0,   t2,    t1   #a = temp2 – temp1
```

$\downarrow$ Ordering of instructions matters (must follow the order of operations)

The Zero Register

- Zero appears so often in code and is so useful that it has its register!
- Register zero (x0 or zero) always has the value 0 and cannot be changed!
  - Any instruction with x0 as dst has no effect
- Example uses
  - Assume s1->a, s2->b, s3->c
  - add s3, x0, x0   # c= 0
  - add s1, s2, x0 # a = b

No-Op

- A No-op is an instruction
  - Does not perform any operation
  - Expend to ‘ADDI x0, x0, 0’
  - Still takes some time to process this no-op instruction (why?)
    - Instruction fetch and decode
  - Why does the processor need this no-op instruction?
    - Sync. I/O devices
      - Force the CPU to wait a little for external devices to complete their work and report data to the CPU
    - Padding in the data alignment

Summary

- Instruction set architecture (ISA) specifies the set of commands (instructions) a computer can execute
- CISC processor reduces the number of instruction count
- RISC processor breaks complex instructions into multiple simple ones
- RISC-V employs load-store architecture
  - Arithmetic and logic instructions working on registers only
  - Specific instructions to move data from and to memory

Takeaway Questions

- What is the value of RISC-V Register 1 (x1 = x0 + x0)?
  - (A) 1
  - (B) 0
  - (C) 2
- What are advantages of the RISC instructions?
  - (A) Reducing the complexity of the processor
  - (B) Decreasing the number of executed instructions
  - (C) Simplify the compiler design

Outline

- RISC vs. CISC
- RISC-V Registers
- Basic Arithmetic Instructions
- Immediates
- Data Transfer Instructions

Immediates

- Numerical constants are called immediates

  opi dst, src, imm

  - Operation names end with “I”, replace second source register with an immediate
- Ex: add immediate:

  addi x3, x4, -10 (in RISC-V)

  f = g – 10 (in C)

  where RISC-V registers x3, and x4 are associated with C variables f, g

Immediates & Sign Extension…

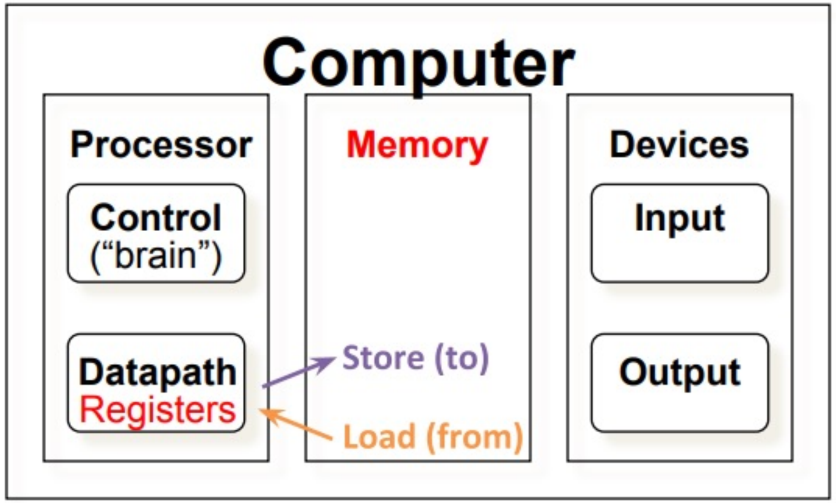
- Immediates are small
    - An I-type instruction can only have 12 bits of immediate
    - The immediate value is a 12-bit signed number, ranging from $-(2^{n-1})$ -2048 to $(2^{n-1} - 1)$ -2048
    - 1 sign bit and 11 value bits
- In RISC-V, immediates are "sign extended"
    - addi   rd, rs, simm12
      sign extend 12-bit immediate to 32-bit and do "add" with the register "rs". The result is written back to the register "rd"
    - Sign extension of -2047 decimal (MSB =1)
      1000 0000 0000 -> 1111 1111 1111 1111 1111 1000 0000 0000

Outline

- RISC vs. CISC
- RISC-V Registers
- Basic Arithmetic Instructions
- Immediates
- Data Transfer Instructions

Five Components of a Computer

- Data transfer instructions are between registers (Datapath) and memory
  - Allow us to fetch and store operands in memory



Data Transfer

- C variables map onto registers;
- What about large data structures like the array?
  - Our one-dimensional array indexed by addresses starting at 0
- RISC-V instructions only operate on registers!
- Data transfer instructions move data between register and memory
  - Store: register TO memory
  - Load: register FROM memory

Data Transfer

- Instruction syntax for data transfer

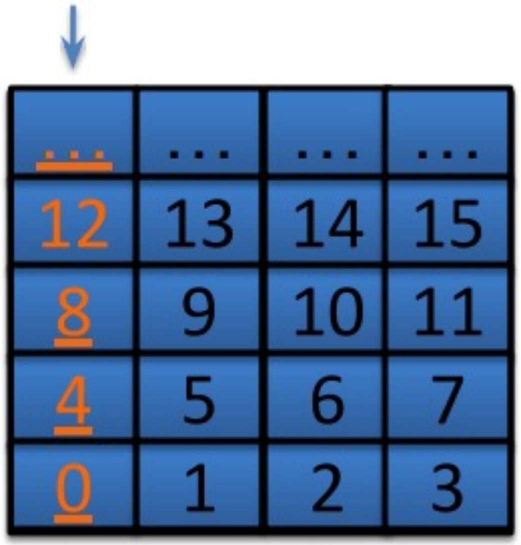
  memop reg, off(bAddr)

  - memop = operation name (“operator”)
  - reg = register for operation source or destination
  - bAddr = register with a pointer to memory (“base address”)
  - off = address offset (immediate) in bytes (“offset”)
  - Access memory at address bAddr + off
  - A register holds a word of raw data (no type)
  - Make sure to use a register (and offset) that points to a valid memory address

Memory is Byte-Addressed

- What was the smallest data type in C?
  - A char, which was a byte (8 bits)
  - Everything is multiples of 8 bits (e.g. 1word = 4 bytes)
  - Memory addresses are indexed by bytes, not words
- Word addresses are 4 bytes apart
  - Word addr is same as left-most byte
  - Addrs must be multiples of 4 to be “word-aligned”
- Pointer arithmetic not done in assembly

Assume here addr of lowest byte in word is addr of word



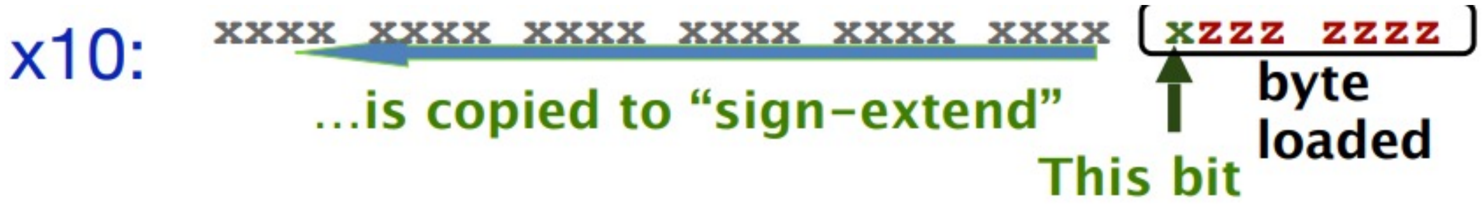
Data Transfer Instruction

- Load Word (lw)
  - Takes data at address bAddr+off FROM memory and place it into reg
- Store Word (sw)
  - Takes data in reg and stores it TO memory at address bAddr+off
- Example usage

```
# addr of int A[] -> s3, a -> s2
lw    t0,12(s3) # $t0=A[3]
add   t0,s2,t0 # $t0=A[3]+a
sw    t0,40(s3) # A[10]=A[3]+a
```

Data Transfer Instruction

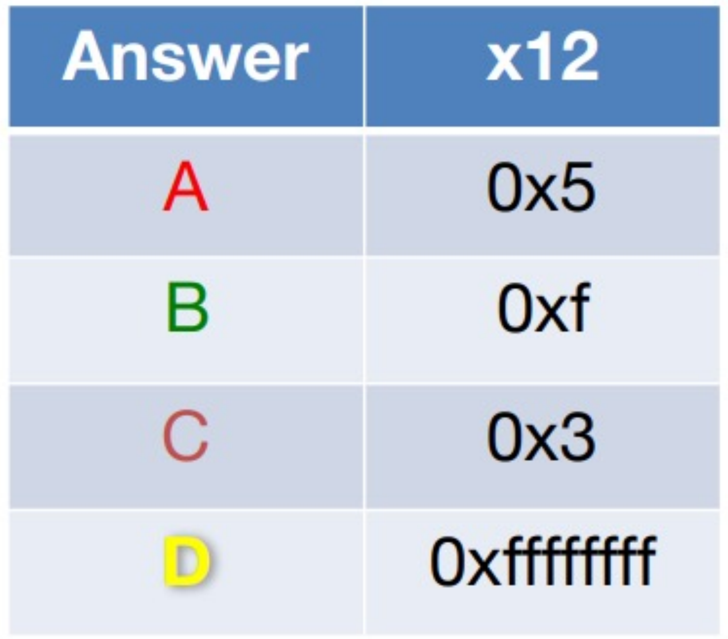
- RISC-V has byte data transfers
  - Load byte: lb
  - Store byte: sb
- Example
  - lb x10, 1(x11)
  - Copies 8 bits (Byte) and make sign-extend to 32-bit than write to rd


x10:
xxxx xxxx xxxx xxxx xxxx xxxx xzzz zzzz
...is copied to "sign-extend"
This bit
byte loaded

Loading and Storing Bytes

- RISC-V has **byte** data transfers:
  - Load byte: lb
  - Store byte: sb
- For example

  -
    ```
    addi    x11, x0, 0x3f5
    sw      x11, 0(x5)
    lb      x12, 1(x5)
    ```
  - What is the value in x12?
    - Note that 0x3f5 (HEX) = 0011 1111 0101(BIN)
      3 f 5
      0x3f5 = 1013(DEC)



Answer	x12
A	0x5
B	0xf
C	0x3
D	0xffffffff

Loading and Storing Bytes

- RISC-V is “little-endian”
  - Byte[0] = least significant byte of the number
  - Byte[3] = most significant byte of the number
- For this example
  - Byte[0] = 0xf5
  - Byte[1] = 0x03
  - Byte[2] = 0x00
  - Byte[3] = 0x00

Summary

- Hardware registers provide a few very fast variables for instruction to operate on
- Assembly code is human-readable version of computer’s native machine code, converted to binary by an assembler

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

The range of the 12-bit signed immediate is -2^{12} <-> $2^{12} - 1$

Sign

$\boxed{1}$000 0000 0000 ⇔ 1111 1111 1111

-2048(DEC) ⇔.   2047(DEC)

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

Sign

0x8f5 <=> $\boxed{1}$000 1111 0101 (2' complement) <=> -779(DEC)

1000 1111 0101 (2'complement) -> -779
1000 1111 0100 (1' complement)
0111 0000 1011 (unsigned 779)

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12. 1(x5)
```

Sign

0x8f5 <=> $\boxed{1}$000 1111 0101 (2' complement) <=> -779(DEC)

1111 1111 1111 1111 1111 1000 1111 0101 (Signed extend 0x8f5 to 32-bits) => 0xfffff8f5

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi   x11, x0, 0x8f5
sw     x11, 0(x5)
lb     x12. 1(x5)
```

- **addi x11, x0, 0x8f5**
- The immediate got sign extended, x11 is 0xfffff8f5 because x11 is signed 32-bit register
- **sw x11, 0(x5)**
- the value of x11 is copied to x5 = 0xfffff8f5

Takeaway Questions

- What is the value in x12?
  - (A) 0x8
  - (B) 0xf8
  - (C) 0xfffffff8

```
addi    x11, x0, 0x8f5
sw      x11, 0(x5)
lb      x12, 1(x5)
```

- **lb x12, 1(x5)**
- Load byte sign extend to the register
- 0(x5) = 0xf5
- 1(x5) = 0xfffffff8