L3_1 ISAs – Instructions and Memory

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- Identify the addressing modes of memory operations used in assembly-language instructions and programs
- Understand encoding of addressing for assembly-language instructions for load, store, and branching instructions
- Usage and encoding of labels for assembly-language programs

Resources

- Many resources on 370 website
  - https://www.eecs.umich.edu/courses/eecs370/eecs370.f20/resources/
    - ARMv8 references
    - Binary, Hex, and 2's compliment
- Discussion recordings
- Piazza
- Office hours

What is a Bit?

- Bit: Smallest unit of data storage
  - Values `[0, 1]`
  - Many things will be measured (for size) in bits
    - 32-bit register – a register with 32 binary digits of storage capacity
    - 32-bit instruction – machine code instruction that has 32 binary digits, i.e., an unsigned integer in the range `0` to `2^32` (`0` to `4,294,967,296`)
    - 32-bit address – memory addresses with 32 binary digits
    - 32-bit operating system – computer with 32-bit addresses
- Byte: A collection of 8 bits (contiguous)
  - On many computers, the granularity for addresses
- Word: natural group of access in a computer
  - Usually 32 bits
  - Useful because most data exceeds 1 byte of storage need

Assembly and Machine Code

Review! Example ISA

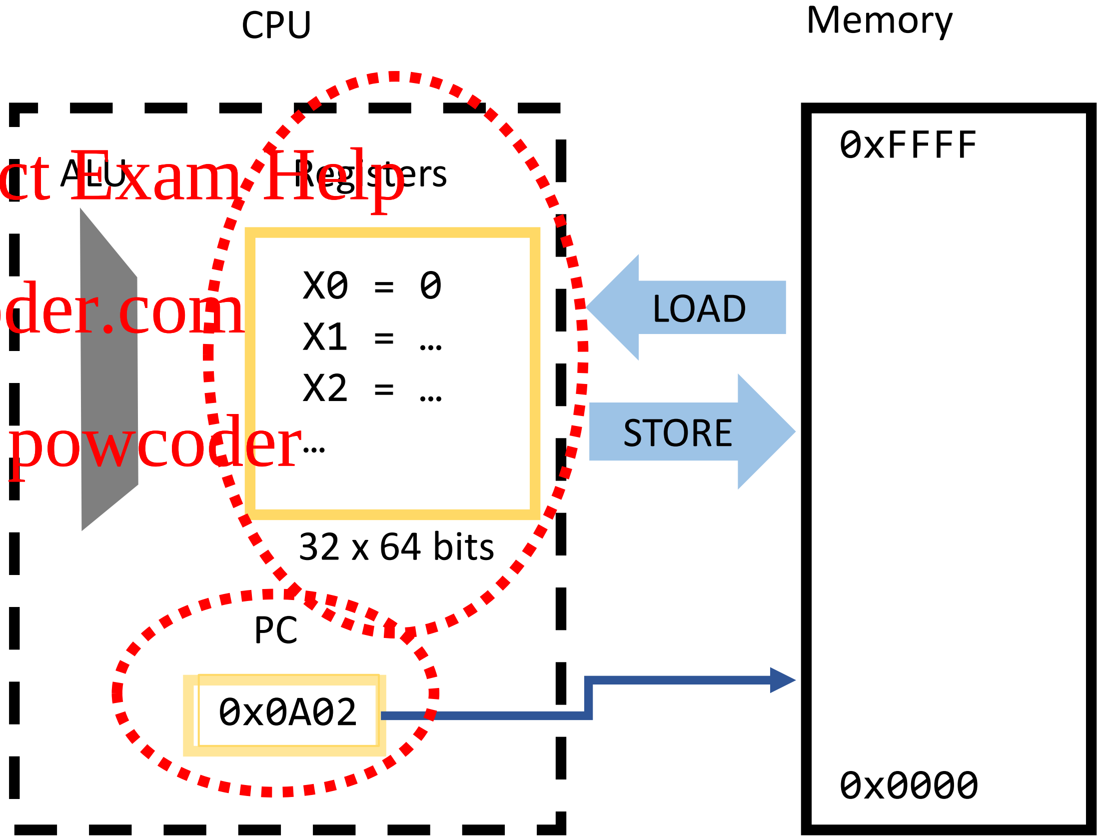
- von Neumann architecture: computers store data and instructions in the same memory
- Instructions *are* data, *encoded* as a **number**

	opcode	dest	src1	src2
Assembly code	ADD	X2	X3	X1
Machine code	011011	010	011	001

Registers

Review! Example ISA

- Registers
  - Small array of storage locations in the processor – general purpose registers
  - Part of the processor – fast to access
  - Direct addressing only
    - That means they can not be accessed by an offset from another address
  - Special purpose registers
    - Examples: program counter (PC), instruction register (IR)



Registers

- ARMv8
  - We will use LEGv8 from Patterson & Hennessy textbook
  - 32 registers, X0 through X31
  - 64-bit wide (64 bits of storage for each register)
  - Some have special uses, e.g., X31 always contains the value 0

- LC-2K
  - Architecture used in course projects
  - 8 registers, 32 bits wide each

LC2K is same as LC-2K
Appears both ways in documents in 370

Special Purpose Registers

- Return address
  - Example: ARM register X30, also known as Link Register (LR)
  - Holds the return address or link address of a subroutine

- Stack pointer
  - Examples: ARM register X28 – SP, or x86 ESP

  - Holds the memory address of the stack

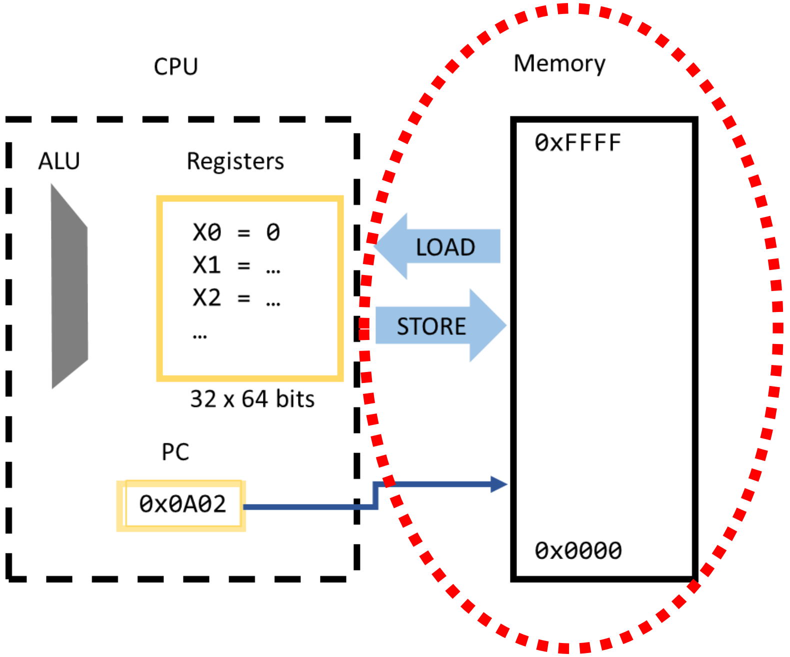
- Frame pointer
  - Example: ARM register X29 – FP
  - Holds the memory address of the start of the stack frame
- Program counter (usually referred to as PC)
  - Cannot be accessed directly in most architectures
    - This would be a security problem!

These registers store memory addresses

Special Purpose Registers

- `0` value register (ARM register X31 – XZR)
  - no storage, reading always returns `0`
  - lots of uses – ex: mov→add
- Status register
  - Examples: ARM SPSR, or x86 EFLAGs
  - Status bits set by various instructions
    - Compare, add (overflow and carry) etc.
  - Used by other instructions like conditional branches

Memory Storage

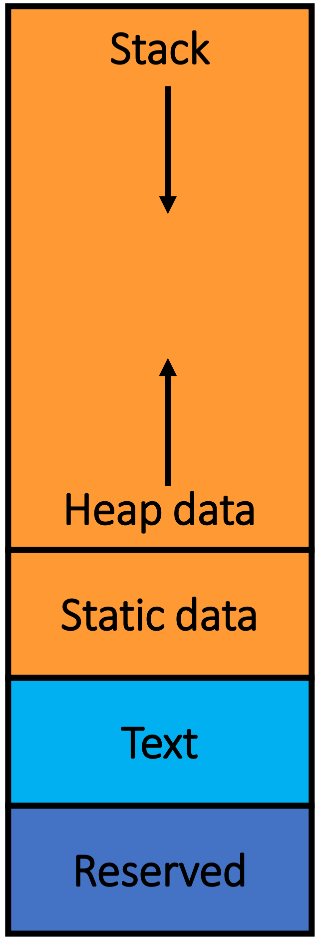


- Large array of storage accessed using memory addresses
- A machine with a 32-bit address can reference memory locations 0 to $2^{32}-1$ (or 4,294,967,295).
- A machine with a 64-bit address can reference memory locations 0 to $2^{64}-1$ (or 18,446,744,073,709,551,615—18 exa-locations)
    - In practice 64-bit machines do not have 64-bit physical addresses

Assembly instructions have multiple ways to access memory (i.e., addressing)

Memory: ARM (Linux) Memory Image

Review!



Activation records: local variables, parameters, etc.

Dynamically allocated data—new or `malloc()`

Global data and static local data

Machine code instructions (and some constants)

Reserved for operating system

Addressing Modes

- Addressing (accessing memory using addresses) modes for assembly instructions
  - Direct addressing – memory address is in the instruction
  - Register indirect – memory address is stored in a register
  - Base + displacement – register indirect plus an immediate value
  - PC-relative – base + displacement using the PC special-purpose register

Direct Addressing

Example ISA (Simplified)

- Specify the address as immediate (constant) in the instruction

```
load  r1, M[ 1500 ]   ; r1 <- contents of location 1500
jump #6000            ; jump to address 6000
```

Direct Addressing

Example ISA (Simplified)

- Specify the address as immediate (constant) in the instruction

```
load  r1, M[ 1500 ]   ; r1 <- contents of location 1500
jump #6000            ; jump to address 6000
```


- Not practical for something like ARMv8


```
load  r1, M[1073741823]     // 1073741823 is the address in memory
```

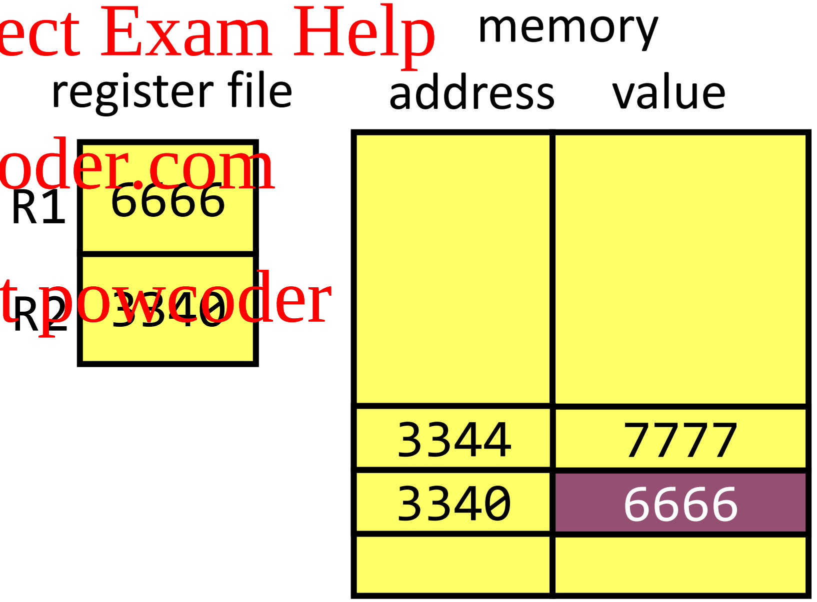
With 32-bit instruction encodings, a 32-bit address would fill the instruction!

Register Indirect

Example ISA (Simplified)

- Store reference address in a register


```
load r1, M[ r2 ]
add  r2, r2, #4
load r1, M[ r2 ]
```



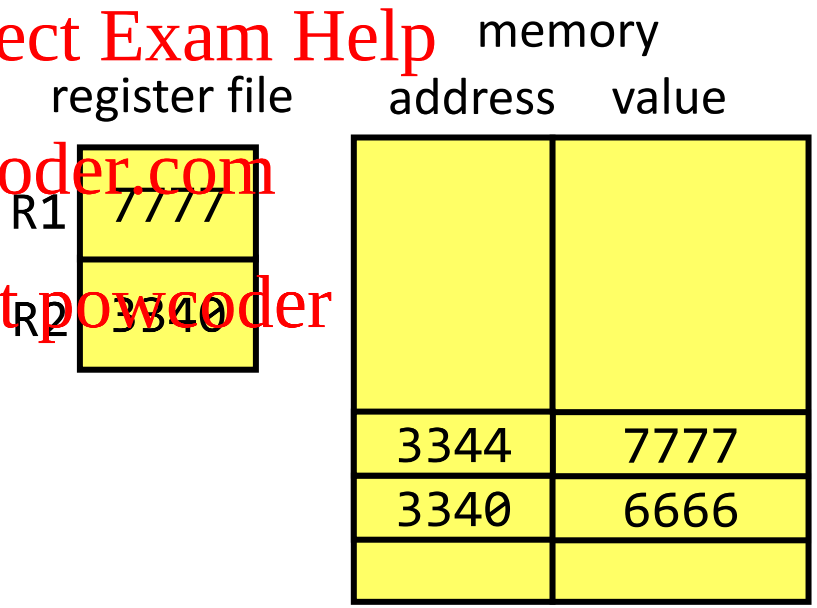
Useful for pointers and arrays
`load r1, M[ r2 ]` is a pointer dereference in assembly

Example ISA (Simplified)

Base + Displacement

- Most common addressing mode
- Address is computed as register value + immediate


```
load r1, M[ r2 + 4 ]
```



Useful for accessing structures or class objects

C code

```
struct Distance {
  int feet;
  int inch;
} x, y;
```

```
x.feet = 11;
y.inches = 5;
```

Addressing Example 1

Example ISA (Simplified)

What are the contents of registers and memory after executing the assembly instructions?

```
load  r2, M[ r3 ]
load  r3, M[ r2 + 4 ]
store r3, M[ r2 + 8 ]
```


register file

R1	
R2	
R3	

memory

address	value
108	
104	
100	

Starting values

register file

R1	0
R2	10
R3	108

memory

address	value
108	100
104	-1
100	108

Addressing Example 1

Example ISA (Simplified)

What are the contents of registers and memory after executing the assembly instructions?

```
1 load  r2, M[ r3 ]        104
2 load  r3, M[ r2 + 4 ]
3 store r3, M[ r2 + 8 ]    108
```


register file

R1	0
R2	100
R3	-1

memory

address	value
108	-1
104	-1
100	108

Starting values

register file

R1	0
R2	10
R3	108

memory

address	value
108	100
104	-1
100	108

Addressing Example 1

Example ISA (Simplified)

What are the contents of registers and memory after executing the assembly instructions?

```
load  r2, M[ r3 ]
load  r3, M[ r2 + 4 ]
store r3, M[ r2 + 8 ]
```

```
load  r2, M[ r3 ]
```


register file

R1	0
R2	100
R3	108

memory

address	value
108	100
104	-1
100	108

Starting values

register file

R1	0
R2	10
R3	108

memory

address	value
108	100
104	-1
100	108

Addressing Example 1

What are the contents of registers and memory after executing the assembly instructions?

```
load  r2, M[ r3 ]
load  r3, M[ r2 + 4 ]
store r3, M[ r2 + 8 ]
```

Example ISA (Simplified)

```
load  r3, M[ r2 + 4 ]
```


register file

R1	0
R2	100
R3	-1

memory

address	value
108	100
104	-1
100	108

Starting values

register file

R1	0
R2	10
R3	108

memory

address	value
108	100
104	-1
100	108

Addressing Example 1

Example ISA (Simplified)

What are the contents of registers and memory after executing the assembly instructions?

```
load  r2, M[ r3 ]
load  r3, M[ r2 + 4 ]
store r3, M[ r2 + 8 ]
```

```
store r3, M[ r2 + 8 ]
```


register file

R1	0
R2	100
R3	-1

memory

address	value
108	-1
104	-1
100	108

Starting values

register file

R1	0
R2	10
R3	108

memory

address	value
108	100
104	-1
100	108

Program Counter (PC) Relative

Example ISA (Simplified)

- Useful for project - P1a
- Variation of base + displacement
- PC register is the base

Useful for branch instructions!

Relative distance from PC can be positive or negative

```
jump   [ -8 ]

jump   [ 2 ]
```

PC-Relative Addressing

LC-2K ISA

- Machine language instructions (encoded from an assembler) use numbers for pc-relative addressing'
- Assembly language instructions (written by people) use ***labels***

PC-Relative Addressing

LC-2K ISA

- Machine language instructions (encoded from an assembler) use numbers for pc-relative addressing'
- Assembly language instructions (written by people) use ***labels***


```
Address
  0           lw 0 1 five     load reg1 with 5 (symbolic address)
  1           lw 1 2 3        load reg2 with -1 (numeric address)
  2   start   add 1 2 1       decrement reg1
  3           beq 0 1 2       goto end of program when reg1==0
  4           beq 0 0 start   go back to the beginning of the loop
  5           noop
  6   done    halt            end of program
  7   five    .fill 5
  8   neg1    .fill -1
  9   stAddr  .fill start     will contain the address of start (2)
```

PC-Relative Addressing

LC-2K ISA

```
Address
      0            lw 0 1 five      load reg1 with 5 (symbolic address)
      1            lw 1 2 3         load reg2 with -1 (numeric address)
      2    start   add 1 2 1        decrement reg1
      3            beq 0 1 2        goto end of program when reg1==0
      4            beq 0 0 start    go back to the beginning of the loop
      5            noop
      6    done    halt             end of program
      7    five    .fill 5
      8    neg1    .fill -1
      9    stAddr  .fill start      will contain the address of start (2)
```

PC-Relative Addressing

beq 0 0 start

2

$2 = 4 + 1 + \text{OFFSET}$

$\boxed{-3 = \text{OFFSET}}$

COMMENTS

```
Address
      0            lw 0 1 five        load reg1 with 5 (symbolic address)
      1            lw 1 2 3           load reg2 with -1 (numeric address)
      2   start    add 1 2 1          decrement reg1
      3            beq 0 1 2          goto end of program when reg1==0
      4            beq 0 0 start      go back to the beginning of the loop
      5            noop
      6   done     halt               end of program
      7   five     .fill 5
      8   neg1     .fill -1
      9   stAddr   .fill start        will contain the address of start (2)
```


beq 0==1, $\text{PC} + 1 + \text{OFFSET}$

$3 + 1 + 2 = \boxed{6}$

Project P1a

- After reading specification, downloading starter files, creating project…

- Write test cases to verify your C code
  - Test cases written in LC-2K assembly

- Recommended for a start:


```
t0.ac: halt
t1.ac: noop
       halt
t2.ac: add 1 2 3
       halt
```

```
t3.ac: nor 3 1 4
       halt
```

Logistics

- This is the first of 3 videos for lecture 3
  - L3_1 – ISAs – Instructions and Memory
  - L2_2 – Two's Complement
  - L2_3 – LC-2K ISA
- There are two worksheets for lecture 3
  1. Addressing and 2's complement
  2. LC-2K program encoding
- Move on to L3_2 when ready