CS061:
Machine Organization & Assembly Language
Lab 2

Agenda

1. Presentation:
    a. Branches Revisited
    b. Loads/Stores
    c. ASCII
    d. Lab Descriptions

2. Work Time / Questions / Demos

Branches Revisited

```
DO_WHILE_LOOP
    ADD R1, R1, #-1
    BRp DO_WHILE_LOOP
END_DO_WHILE_LOOP
```

- 3 special registers:
  - N, Z, P

N	Z	P
0	0	1

- Result of ADD sets one of those to 1.
  - Other registers are zero.
  - E.g. if result of ADD is positive, P register set to 1 (rest set to 0).
- BR checks those condition registers.
  - If “BRp” and P register set, then jumps back to “DO_WHILE_LOOP.
  - Otherwise, “BRp” does nothing and program continues to “END_DO_WHILE_LOOP”.

Loads/Stores

- 3 mains types of Loads and Stores
  - Load Direct (LD) and Store Direct (ST)
  - Load Indirect (LDI) and Store Indirect (STI)
  - Load Relative (LDR) and Store Relative (STR)

- *What about LEA?*
  - LEA isn’t really a load (doesn’t read from memory)
  - Puts a memory address (represented by a label) into a register

Load Direct

```
        .ORIG x3000

           Destination   Label
x3000   LD R1, COOKIES

x3001   HALT

x3002   COOKIES .FILL    #3

        .END
```

- LD (Load Direct) reads value from memory address and stores it at register.
    - Reads value from address x3002 (named by “COOKIES”) and stores it in R1.
    - R1 <- Mem [“COOKIES” = x3002].
    - R1 is then 3.

Address	Value
x3000	… (LD R1, COOKIES)
x3001	… (HALT)
x3002	$\boxed{\#3}$

COOKIES

Registers	Value
R1	$\boxed{3}$

Store Direct

```
.ORIG x3000

x3000   AND R1, R1, #0
x3001   ADD R1, R1, #5
             Source    Label
x3002   ST R1, COOKIES

x3003   HALT

x3004   COOKIES .FILL   #3

.END
```

- ST (Store Direct) puts value from register at a memory address.
  - Stores value in R1 (5) at memory address x3004 (named by “COOKIES”).
  - Mem[“COOKIES” = x3004] = R1
  - Value at x3004 is now 5.

Registers	Value
R1	5

	Address	Value
	x3002	… (ST R1, COOKIES)
	x3003	… (HALT)
COOKIES	x3004	5

Cookies Out of Reach

```
       .ORIG x3000
x3000  LD R0, COOKIES
x3001  HALT
       .END
       .ORIG x4000
x4000  COOKIES .FILL   #2
       .END
```

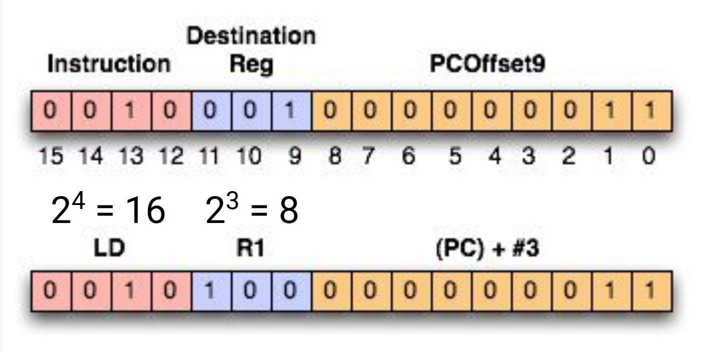
- What happens when we assemble this?
- Gets:

```
error: cannot encode as 9-bit 2's complement number
```

Cookies Offset

```
      .ORIG x3000
x3000 LD R0, COOKIES
x3001 HALT
      .END

      .ORIG x4000
x4000 COOKIES .FILL   #2
      .END
```



- LD encodes offset between current instruction (PC) and target address (represented by label)
  - x4000 - x3000 = x1000
  - x1000 = 1000000000000
  - Cannot encode x1000 in 9-bits

Load Indirect (LDI)

```
        .ORIG x3000
              Destination      Label
x3000   LDI R1, COOKIES_ADDR
x3001   HALT
x3002   COOKIES_ADDR .FILL  x4000
        .END
        .ORIG x4000
x4000   COOKIES .FILL   #4
        .END
```

- Use a layer of indirection to access the value.
- **LDI** (Load Indirect) reads the value at a memory address (label) and *uses that as an address* to load a value into a register.
  - LDI first reads value at label “COOKIE_ADDR” which is x4000.
  - Uses that as an address and then reads the value at x4000 and loads that value (4) into R1.
  - R1 <- Mem[Mem[COOKIE_ADDR]]

Registers	Value
R1	4

	Address	Value
	x3000	… (LDI R1, COOKIES_ADDR)
COOKIES_ADDR	x3003	x4000
COOKIES	x4000	4

Store Indirect (STI)

```
;  Source            Label
x3002   STI R1, COOKIES_ADDR

x3003   HALT

x3004   COOKIES_ADDR .FILL   x4000

        .END

        .ORIG x4000

x4000   COOKIES .FILL   #1

        .END
```

- **STI** (Store Indirect) stores register value at a memory address specified by the value at another memory address (label).
  - STI reads value at label “COOKIE_ADDR” (x4000).
  - Uses that as an address and stores value of R1 (5) at x4000.

Registers	Value
R1	5

Label	Address	Value
	x3002	… (STI R1, COOKIES_ADDR)
COOKIES_ADDR	x3004	x4000
COOKIES	x4000	5

Load Relative (LDR)

```
.ORIG x3000

LD R0, COOKIES_ADDR
  Destination      Offset
LDR R1, R0, #0
        Addr Reg

HALT

COOKIES_ADDR .FILL  x4000

.END

.ORIG x4000

COOKIES .FILL   #7

.END
```

- Similar to LDI/STI
  - Register has address instead of label.
  - R0 has value x4000.
- Format: LDR/STR <Source/Dest Register> <Address Register> <Offset>

- **LDR** (Load Relative) loads value into a register (R1) from memory address specified by a register (R0).
  - R1 <- Mem[R0 + 0]

Registers	Value
R0	x4000
R1	7

	Address	Value
	x3001	… (LDR R1, R0, #0)
COOKIES_ADDR	x3003	x4000
COOKIES	x4000	7

Store Relative (STR)

```
; ...
LD R0, COOKIES_ADDR
STR R1, R0, #0
HALT
COOKIES_ADDR .FILL  x4000
.END
.ORIG x4000
COOKIES .FILL  #7
.END
```

Destination

Offset

Addr Reg

- **STR** (Store Relative) stores a value from a register (R1) to a memory address specified by a register (R0).
  - Mem[R0 +0] <- R1

Registers	Value
R0	x4000
R1	5

	Address	Value
	x3003	… (STR R1, R0, #0)
COOKIES_ADDR	x3005	x4000
COOKIES	x4000	5

ASCII & Characters

- ASCII is a mapping between numbers and characters.
    - E.g. #48 represents ‘0’
    - E.g. #65 represents ‘A’

- **OUT** (Trap x21) prints out the ASCII character representation of the value in *R0*.
    - E.g. if R0 has value #70 (decimal 70), OUT would print out “F” to console.
    - Use an ASCII table to see the mapping between values and characters.

48	0	64	@	80	P	96	`	112	p
49	1	65	A	81	Q	97	a	113	q
50	2	66	B	82	R	98	b	114	r
51	3	67	C	83	S	99	c	115	s
52	4	68	D	84	T	100	d	116	t
53	5	69	E	85	U	101	e	117	u
54	6	70	F	86	V	102	f	118	v
55	7	71	G	87	W	103	g	119	w
56	8	72	H	88	X	104	h	120	x
57	9	73	I	89	Y	105	i	121	y
58	:	74	J	90	Z	106	j	122	z
59	;	75	K	91	[	107	k	123	{
60	<	76	L	92	\	108	l	124	|
61	=	77	M	93	]	109	m	125	}
62	>	78	N	94	^	110	n	126	~
63	?	79	O	95	_	111	o	127	DEL

Exercise 1

- Use '.FILL' pseudo-op to create two labels (DEC_65 and HEX_41) with the values #65 and x41 respectively.
    - Check the binary values of these two numbers - what do you notice?
    - Use an ASCII table to see what these values represent when interpreted as characters rather than numbers.

- Use LD to load values into registers R3 and R4 respectively.

Exercise 2/3

- Move data far away (i.e. at x4000 & x4001) and load/store the values into/from registers.

- Tips:
  - Review how LDI and STI work (both add a layer of indirection onto a normal load/store) for exercise 2.
  - Review how LDR and STR work for exercise 3.
  - Remember to increment both R3 and R4 before storing them back.

Exercise 4

- Use a loop to print out ASCII characters.
    - The ASCII table is a mapping from values to characters.

- Counter-Controlled Loop: Using a register as a counter and repeating the loop while that register is greater than zero.

- **OUT** (Trap x21) prints out the ASCII character representation of the value in *R0*.
    - E.g. if R0 has value #33 (decimal 33), OUT would print out “!” to console.
    - Use an ASCII table to see the mapping between values and characters.

Demo Info

- **Please sign up to demo only when you have completed all exercises and fully understand your code.**

- Lab Grade Breakdown:
    - 3 points for attendance.
    - 7 points for demoing (+1 bonus point).

- 1 bonus point in the demo category if lab is demo'd *before/during* Friday.
- 3 point penalty if lab is demo'd during the next lab session.