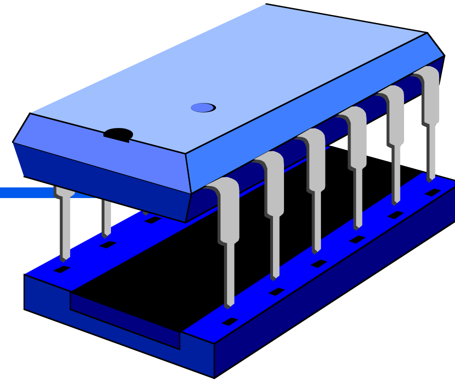

TOY1 Assembly Programming

Professor Kin K. Leung

kin.leung@imperial.ac.uk

www.commsp.ee.ic.ac.uk/~kkleung/

Heavily based on materials by Dr. Naranker Dulay

TOY1 Instruction Set

Opcode	Assembly Instruction	Action
0000	STOP	Stop Program Execution
0001	LOAD Rn, [Addr]	Rn = Memory [Addr]
0010	STORE Rn, [Addr]	Memory [Addr] = Rn
0011	ADD Rn, [Addr]	Rn = Rn + Memory [Addr]
0100	SUB Rn, [Addr]	Rn = Rn - Memory [Addr]
0101	GOTO Addr	PC = Addr
0110	IFZER Rn, Addr	IF Rn == 0 THEN PC = Addr
0111	IFNEG Rn, Addr	IF Rn < 0 THEN PC = Addr

Example 1 - Multiplication

- Compute $A = B * C$

$$B * C = \sum_{N=1}^{C} B$$

- Examples $12 * 3 = 12 + 12 + 12$

$12 * 1 = 12$

$12 * 0 = 0$

A Solution

```
; Given:        A, B, C
; Pre:          C >= 0
; Post:         A = B * C
sum = 0         ; accumulate result in sum
n = C           ; indicates how many additions remain
loop
    exit when n <= 0       ; no more additions remain
    sum = sum + B          ; add another B
    n = n - 1              ; one less addition to do
end loop
A = sum
```

Variables

- We'll allocate variables to main memory as follows:

A	to	Memory [100H]
B	to	Memory [101H]
C	to	Memory [102H]
sum	to	R1
n	to	R2

sum = 0

- We need to perform R1 = 0 but the closest instruction we have is **LOAD Rn, [Addr]**

- So we'll pre-set a memory location (e.g. memory word 200H) to 0.

- Now sum = 0 simply translates to

  LOAD R1, [200H] ; sum = 0

Memory Contents

- We'll fill main memory as follows:

Address	Contents
➢ 100H	A
101H	B
102H	C
➢ 200H	0
➢ 080H	1st Instruction of program
081H	2nd Instruction of program
	Etc.

n = C

- This is easy to translate **LOAD R2, [102H]**

- We now have

```
080H    LOAD  R1, [200H]   ; sum = 0
081H    LOAD  R2, [102H]   ; n = C
...
100H    A                  ; holds A
101H    B                  ; holds B
102H    C                  ; holds C
...
200H    0                  ; constant 0
```

exitloop when n <= 0

➤ Let's consider a simpler statement **exitloop when n = 0**

Statement	**Address**	**Instructions**
loop	L0	
exit when n = 0		IFZER R2, LY
instructions	...	...
end loop		GOTO L0
....	LY	....

exit when n <= 0

➢

Statement	**Address**	**Instructions**
loop		
exit when n <= 0	L0	IFZER R2, LY IFNEG R2, LY
instructions	...	...
end loop		GOTO L0
....	LY	....

Memory Contents

```
080H    LOAD  R1, [200H]     ; sum = 0
081H    LOAD  R2, [102H]     ; n = C
082H    IFZER R2, LY         ; loop exit when n <= 0
083H    IFNEG R2, LY         ;
...     ...                  ; ...
...     GOTO 082H            ; end loop
LY      ...                  ; ...
...     ...                  ; ...
100H    A                    ; holds A
101H    B                    ; holds B
102H    C                    ; holds C
...
200H    0                    ; constant 0
```

sum = sum + B

- This simply translates to

ADD R1, [101H]

n = n - 1

- This is easy to translate if we pre-set another location (memory word 201H) with the constant 1

SUB R2, [201H]

Final Program

```
080H        LOAD  R1, [200H]      ; sum = 0
081H        LOAD  R2, [102H]      ; n = C
082H        IFZER R2, 087H        ; loop exit when n <= 0
083H        IFNEG R2, 087H        ;
084H        ADD   R1, [101H]      ; sum = sum + B
085H        SUB   R2, [201H]      ; n = n - 1        ...
086H        GOTO  082H            ; end loop
087H        STORE R1, [100H]      ; A = sum
088H        STOP                  ; That's all folks!
100H        A                     ; holds A
101H        B                     ; holds B
102H        C                     ; holds C
200H        0                     ; constant 0
201H        1                     ; constant 1
```

Example 2 - Vector Sum

- Sum = Memory [200H] + ... + Memory [200H + 99]

```
sum  = 0                                  ; accumulates result
n = 100                                   ; no. of elements to add
addr = 200H                               ; address of 1st element
loop
    exit when n <= 0                      ; elements remaining
    sum = sum + Memory [addr]             ; add next element
    addr = addr + 1                       ; advance to next element
    n = n - 1                             ; one less element to add
end loop
```

TOY1 Instruction Set Continued

Opcode	Assembly Instruction	Action
1000	spare	
1001	LOAD Rn, [Rm]	Rn = Memory [Rm]
1010	STORE Rn, [Rm]	Memory [Rm] = Rn
1011	ADD Rn, [Rm]	Rn = Rn + Memory [Rm]
1100	SUB Rn, [Rm]	Rn = Rn - Memory [Rm]
1101	spare	
1110	spare	
1111	spare	

Instruction Format 2

OPCODE	REGn	REGm	UNUSED
4-bit	2-bit	2-bit	8-bit

- Example: Disassemble the instruction CEAB H

CEAB = 1100 1110 1010 1011

SUB R3 [R2] Unused

Memory Allocation

- We'll allocate variables to memories as follows:

Vector	to	Memory [200H] .. Memory [200H+99]

---

sum	to	R0
n	to	R1
addr	to	R2

Constants	at	Memory [000H] onwards
Program	at	Memory [00FH] onwards

TOY1 Vector Sum Program

sum = 0		; accumulates result
	0FH	LOAD R0, [00H]
	00H	0
	01H	1
n = 100		; no. of elements to add
	10H	LOAD R1, [02H]
	02H	100
addr = 200H		; address of 1st element
	11H	LOAD R2, [03H]
	03H	200H

TOY1 Vector Sum Program Contd.

loop exit when n <= 0 ; elements remaining

```
12H   IFZER R1, 18H
13H   IFNEG R1, 18H
```

sum = sum + Memory [addr] ; add next element

```
14H   ADD   R0, [R2]
```

addr = addr + 1 % addr of next element

```
15H   ADD   R2, 01H
```

n = n - 1 % one less element to add

```
16H   SUB   R1, 01H
```

end loop

```
17H   GOTO 12H
18H   STOP
```

Think About

- Translating High-level Language statements to Assembly Language instructions (Compilation)
- Allocating Variables to Registers and Main Memory
- Branching and Looping
- Indirect Addressing