
ASM Programming

Ch. 2-3

ECE331, Spring 2011



Outline

- Structured programming process
- Program structures
- Looping concept
- Looping constructs
- ASM code examples
- Branch instructions
- Relative address mode
- BRSET/BRCLR instructions
- Instruction timing
- Delay loops
- ASM loop/branch examples



Structured Programming Process

1. Define Problem

-what exactly should your program do?

2. Plan Solution = Develop *Algorithm*

-sequence of computational or logical step to transform given inputs to desired outputs

Options

- top-down: start with *goals*, work down to required *outputs*
- bottom-up: start with required *outputs*, work up to *goals*
- divide & conquer
 - use top-down to break into tasks
 - use bottom-up to solve each task
 - best method for large programs

Approaches for algorithm planning

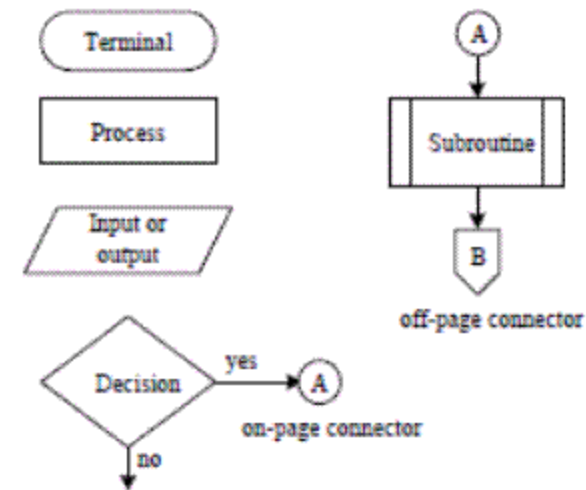
- Pseudocode: non-code list of tasks (example will follow)
- Flowcharts: graphical map of algorithm flow (example follow)

3. Code Algorithm

-ASM, C, higher level language

4. Simulate & Debug

5. Test in Hardware & Debug

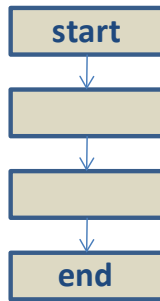


Flow Chart Elements

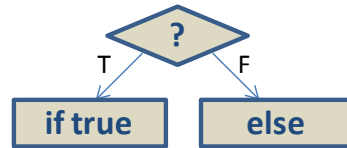


Program Structures

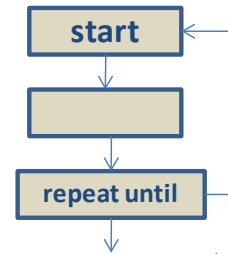
sequential



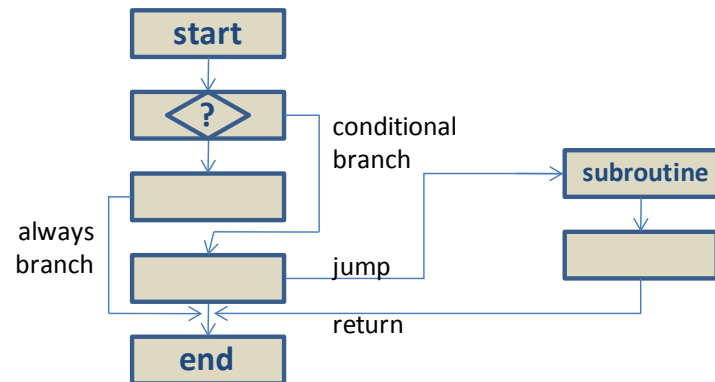
conditional



looping



Assembly Implementations



Looping Concept

CONCEPT: Consider need to sum 10 HW grades

Option 1: "Brute Force"

pseudo code

```
set sum = 0
sum = sum + score 1
sum = sum + score 2
...
sum = sum + score 10
```

ASM code

```
CLA                                ;set sum to zero
ADDA$1000                          ;assume scores saved starting at mem addr $1000
ADDA$1001
...
ADDA$1009
STAA SUM                          ;assume SUM = mem addr to store the sum
```

How many bytes in above ASM code? Approximately $3 \times \text{number_of_scores} = \sim 30$

Code gets longer if we want to add more scores



Looping Concept

Option 2: Use a Loop

pseudo code

```
set sum = 0
set count = 0
while count < 10
    sum = sum + score(count)
    increment count
(loop back to "while")
```

ASM code

See "Simple Looping Code"

Can you identify the loop in this program?

How many bytes of code in the loop?

21, for an infinite number of scores

Simple Looping Code

.ASM code file, text

```
; Loop example for Ch. 3 notes by A.Mason. Mar 09
; Sums 10 values from memory and store result in SUM. Assumes 10
; values to sum are stored at $1000
; assumes prior sum stored at SUM
```

```

                                ORG     $4000
                                LDX     #$1000 ;set x to fist memory address
                                LDAB    SUM   ;load staring sum into accB
                                LDAA    #$00 ;initialize counter to 0
CHECK  CMPA    #$0A ; ?added all 10?
                                BEQ     DONE ; if yes, done
                                ADDB   0,X  ; if no, add # to SUM
                                INX     ;increment IX
                                INCA    ;increment counter
                                BRA     CHECK ;repeat loop
DONE  STAB    SUM ;store result
SUM   EQU     $4400
                                END
```

.LST compiled file, text

Line	Addr	Op Code	Label	Mnemonic	Operand	Notes
1:						; Loop example for Ch. 3 notes by A.Mason. Mar 09
2:						; Sums 10 values from memory & store result in SUM
3:						; assumes 10 values to sum are stored at \$1000
4:						; assumes prior sum stored at SUM
5:	=00004000			ORG	\$4000	
6:	4000 CE 1000			LDX	#\$1000	;set x to fist memory addr
7:	4003 F6 4400			LDAB	SUM	;load staring sum into accB
8:	4006 86 00			LDAA	#\$00	;initialize counter to 0
9:	4008 81 0A		CHECK	CMPA	#\$0A	; ?added all 10?
10:	400A 27 06			BEQ	DONE	; if yes, done
11:	400C EB 00			ADDB	0,X	; if no, add # to SUM
12:	400E 08			INX		;increment IX
13:	400F 42			INCA		;increment counter
14:	4010 20 F6			BRA	CHECK	;repeat loop
15:	4012 7B 4400		DONE	STAB	SUM	;store result
16:	=00004400		SUM	EQU	\$4400	
17:				END		

.S19 machine code record, binary

```
S0030000FC
S1134000CE1000F644008600810A2706EB00084221
S108401020F67B4400D2
S9030000FC
```

Looping Constructs

For Loops

- execute S $i_2 - i_1 + 1$ times, S = any set of instructions, I = counter
- see flowchart below

For I = i_1 to i_2
Do S

Do While Loops

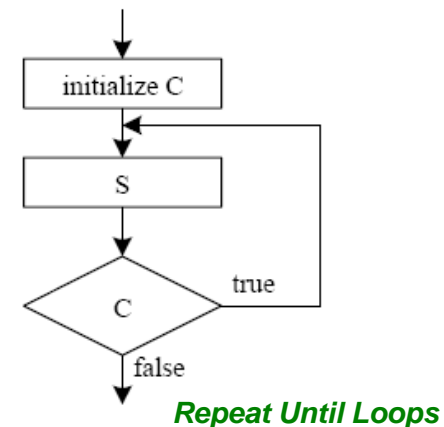
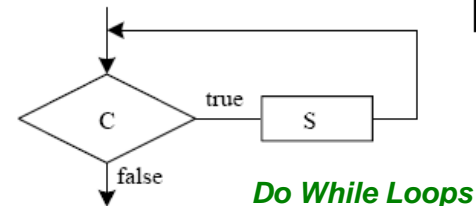
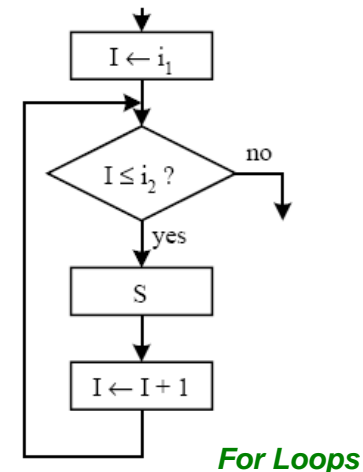
- execute S as long as C is TRUE, C = condition that will be false when done with S
- see flowchart below

While C = true
Do S

Repeat Until Loops

- execute S until C is FALSE, C = condition that will be false when done with S
- functionally similar to Do While but with slightly different flow
- see flowchart below

Repeat S,
Until C = false



ASM Examples

Discuss ASM Examples in HO_5, pg. 4-6

- Simple Arithmetic ASM Program Examples

- Example 1: Write a program to **add** the numbers stored at memory locations \$800, \$801, and \$802, and store the sum at memory location \$900.
- Example 2: Write a program to **subtract** the contents of the memory location at \$805 from the sum of the memory locations at \$800 and \$802, and store the result at the memory location \$900.
- Example 3: Write a program to **add** two 16-bit numbers that are stored at \$800~\$801 and \$802~\$803, and store the sum at \$900~\$901.
- Example 4: Write a program to **subtract** 5 from four memory locations at \$800, \$801, \$802, and \$803.

- Assembly and Execution Example (Chapter 2)

- Also see HO_6 Example ASM Code for more examples



Branch Instructions

- Function of branch instructions
 - implement loops; IF-THEN-ELSE constructs
 - alter execution order of linear instructions
- 68HC12 has 2 types of branches
 - conditional
 - unconditional
 - both use **Relative Addr. Mode**
- **Unconditional Branches**
 - does not depend on any conditions; will always occur
 - **BRA ;branch always**
 - 8-bit signed offset
 - distance to next instr. +127 to -128
 - **EXAMPLE:**
 - ***best to use labels & let assembler determine relative offset operand***



Branch Instructions II

- More unconditional branches

- **LBRA** ; long branch always, 16-bit signed offset
- **JMP** ; jump, 16-bit extended/indexed addr.
 - not relative like BRA; specific 16-bit addr.
 - less compact & more clocks than BRA

- **Conditional Branch**

- branch only if a condition is true
 - conditions derived from **CCR flags**
 - previous instr. must set proper CCR flags
 - can use a Test/Compare instruction to set flags without changing memory values
- **Bxx** ; conditional branches with 8-bit signed offset
- **LBxx** ; conditional long branches with 16-bit signed offset
- all conditional branches use **Relative Addr. Mode**
- ***best to use labels & let assembler determine relative offset operand***



AMS Branch Instructions

From HO_3 (HO_2B Spring 2011)

Unconditional Branches

<i>Mnemonic</i>	<i>Function</i>	<i>Comment</i>
BRN	branch never	useful as delay
BRA	branch always	8-bit signed offset

Simple Conditional Branches

<i>Mnemonic</i>	<i>"if" Function</i>	<i>Condition = Flag</i>
BCC	carry clear	C=0
BCS	carry set	C=1
BEQ	equal	Z=1
BNE	not equal	Z=0
BMI	minus	N=1
BPI	plus	N=0
BVS	overflow set	V=1
BVC	overflow clear	V=0

Unsigned Conditional Branches

<i>Mnemonic</i>	<i>"if" Function</i>	<i>Condition</i>	<i>Flags</i>
BHS	higher or same	$r \geq 0$	C=0
BHI	higher than	$r > 0$	C+Z=0
BLS	lower or same	$r \leq 0$	C+Z=1
BLO	lower than	$r < 0$	C=1

Signed Conditional Branches

<i>Mnemonic</i>	<i>"if" Function</i>	<i>Condition</i>	<i>Flags</i>
BGE	greater or equal	$r \geq 0$	$N \oplus V = 0$
BGT	greater than	$r > 0$	$Z + N \oplus V = 0$
BLE	less or equal	$r \leq 0$	$Z + N \oplus V = 1$
BLT	less than	$r < 0$	$N \oplus V = 1$

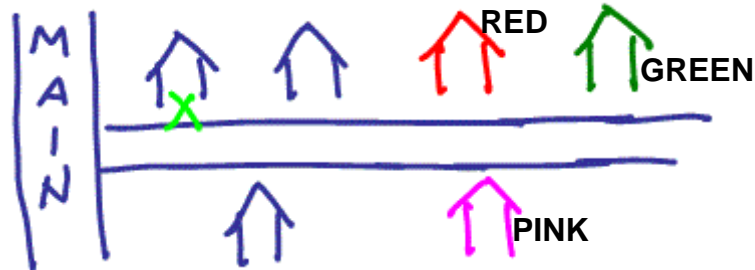
Example:

If all flags = 0 after an instruction, which branches would be taken?



"Relative" Concept

- How can we define the address of the red house?



- Absolute Address

– 3rd House on the left from Main Street

- Relative Address

– locate by offset from reference

– references offsets offset + reference

- Green House
- Pink House
- 'X'



Relative Address Mode

- Function

- adjust PC by value in operand
- operand can be set by program or by assembler
 - assembler can calculate “distance” to a **Label** on line to branch to
- only used in **Branch** instr.

- **EXAMPLES**



Counting Relative Offsets

- **Offset** in Relative Addr. Mode specifies how many instruction bytes (forward or backward) the PC must be moved to get from current PC value to new PC value
 - $PC(next) = PC(current) + Offset$;offset is signed (+/-)
- Number of Bytes per Instruction
 - sum of instr. op-code and operand bytes
 - easy to see in .LST file
- **EXAMPLE: What is the hex value of ??**

<u>Program</u>	<u>op-codes</u>	<u>#bytes</u>
LDDA #10	BC 0A	2
STUK INCA	42	1
STAA \$10B5	A7 01 B5	3
TSTA	97	1
BNE STUK	26 ??	2



BRCLR / BRSET

- Function

- combine test and branch instr.
- can use direct, extended, or indexed addr. mode

- Format

- BRCLR/SET **mem** **mask** **label**

- **mem** = addr. of data to test
- **mask** = bits to check (set to 1 if checked)
- **label** = branch location
- BRCLR: are 1 bits in **mask** = 0 @ **mem**?
 - if yes, branch to **label**
- BRSET: are 1 bits in **mask** = 1 @ **mem**?
 - if yes, branch to **label**

- EXAMPLE **BRCLR 0,X %11110000, NEXT**

- compare value @ <IX> with \$F0
- if bits 7-4 are '0', then branch to NEXT
- else, advance to next instr.



Instruction Timing

- All ASM instructions require a specific amount of time to execute
 - can use this information for program timing, e.g., delay loops

- $T_{instr} \equiv$ Instruction execution time

$$T_{instr} = N \bullet t_{cycle}$$

– $N \equiv$ # clock cycles per instruction

- listed under ~* in textbook Appendix A (HO_4 :Spring 2011 HO_2C)
- N depends on address mode
- **EXAMPLE:**

– $t_{cycle} \equiv$ time of one clock cycle = $1/f_{clk}$, where $f_{clk} \equiv$ clock frequency

- **EXAMPLE:**



Creating Time Delays

- How can a program generate a specific time delay?
 1. Use a hardware timer –will be discussed later
 2. Construct a software delay loop
- Software delay loop

–EXAMPLE:

–What is the loop delay?



Creating Time Delays III

- How can we make a longer loop for longer delay?
 1. More instructions(clocks) within the loop
 2. Use nested loops
- Nested Loop: Loop within a loop
 - EXAMPLE

- Take home exercise:
 - How long is a nested loop delay if $CNTX \& CNTY = \$FFFF$?



ASM Loop Examples

Discuss ASM Loop Examples in HO_5, pg. 7-9

- Example of ASM Code with Branches & Reading Assembled List File (Chapter 2)
 1. Where is the program stored in memory (what addresses)?
 2. What is the op-code for the ASM instruction INCB?
 3. What value is loaded into index in line 9?
 4. What address mode is used to store data to memory in line 12?
 5. What is the value of the relative_address_mode offset byte for BEQ in line 15? Forward or backward?
 6. What is the value of the relative_address_mode offset byte for BRA in line 18? Forward or backward?
 7. What does the program do? Where is the main loop (from what line to what line)?
 8. What is the purpose of line 14?
 9. How many times does copy loop execute? Does the value \$16 get copied?
 10. Could you explain the purpose and operation of each line in this code?
- Simple Looping Code (pg. 8)
- Example Loop using For looping structure (pg. 9)

