Development in Assembler

Design your programs first

• use C or Pascal or pseudo-code
• Translate each construct using templates
• Print out (or view Assembler Listing)
• Single-step your code to validate it

Structured Programming primitives?

• Sequences
• Conditional Statements
• Loops

How are these handled?

Conditionals Jumps

jump instructions that may jump to a destination

eg Jump to location 100 if acc = 0

if acc = 0 then
  pc = 100
else
  pc = pc+1 (eg do next instruction)

These are used to build IF and Switch/Case statements

Most Common 8051 Conditional Jumps

\[ jz \text{ destination} \] jump if Acc = 0
\[ jnz \text{ destination} \] jump if Acc <> 0 (not zero)
\[ jc \text{ destination} \] jump if Carry bit is set
\[ jnc \text{ destination} \] jump if Carry bit is NOT set
\[ djnz \text{ value,dest} \] decrement value and jump if result is not zero

Sequences

Assembler instructions are executed sequentially until explicit jump or subroutine call

⇒ simply have instructions after one another
⇒ no IF statement
⇒ no SWITCH
⇒ no DO

Only unconditional JUMP (or Branch)

or

Subroutine call

or

JUMP if (condition)

Jumps and Branches are two names for the same thing

Never Just One Way

Translate “do X() 10 times” into assembler

count equ 25h ; use mem location 25 for count

Obvious way

\[ \text{mov count,}#0 \] ; Initialise counter
\[ \text{again: lcall X} \] ; Do X
\[ \text{mov a, count} \] ; Get counter to Acc
\[ \text{inc a} \] ; increment it
\[ \text{mov count,a} \] ; store back to mem
\[ \text{clr c} \] ; Clear Carry bit
\[ \text{subb a,}#10 \] ; Count = 10 yet?
\[ \text{jnz again} \] ; if not, repeat

Better way

\[ \text{mov count,}#10 \] ; Initialise counter
\[ \text{again: lcall X} \] ; Do X
\[ \text{dec count} \] ; counter--
\[ \text{mov a, count} \] ; Get counter
\[ \text{jnz again} \] ; if counter is
\[ \text;jnz again} \] ; NOT zero, repeat

Best way

\[ \text{mov count,}#10 \] ; Setup counter
\[ \text{again: lcall X} \] ; Do X()
\[ \text{djnz count,again} \] ; Decrement count
\[ \text{djnz count,again} \] ; repeat if
\[ \text; count not 0? \]
### Conditional Statements

**If** (cond)

true-action

else

false-action

endif

This is usually assembled in the following way:

```
if x = y then
    sub1();  inc(y)
else
    sub2();  dec(x)
endif
```

```
mov acc,x
clr c
subb acc,y ; Acc = X - Y
jnz else ; if Acc<>0 then X<>Y
lcall sub1 ; X=Y so do then part
inc y
sjmp endif
```

```
else: lcall sub2
dec x
endif:
```

Note: Conditional branch is the OPPOSITE of the If test condition

---

### Switch Statement

```
case (ch) {
    case 'A' : { x=1; break; }  
    case 'B' : { x = 2; break; }  
    case 'C' : { x = 3; break; }  
    default :  x = 10;
}
```

```
switch: mov a,ch
    cjne a,'A',isb
    mov x,#1
    sjmp endswitch

    cjne a,'B',isBigB
    sjmp Baction

isBigB: cjne a,'B',default
    ; if 'B' ...

Baction: mov x,#2
    sjmp endswitch

default: mov x,#10 ; else ...
endswitch:
```

Note how execution “falls into” correct code if the condition is false

---

### Subroutine Calls

The **LCALL** opcode stacks the address of the next instruction and then jumps to the specified address

```
main: lcall sub1
lcall sub2
ret
```

```
sub1: stmt1
stmt2
```

```
horrible: stmt3
    ; do NOT do this
    jump into subroutine
stmt4
ret
```

```
sub2: stmt2a
```

```
cjne a,#10,skipit
sjmp horrible
stmt2b
ret
```

---

### Parameters and Return Values

- Pass these in registers - usually Acc & r0 – r7
- can use globals
- more complex CPU would use the stack

---

### Parameters & Return Values

**Simple convention:**

- use accumulator for single parameter
- returns single result in the accumulator
- use r0 & r1 to pass addresses

#### Double Subroutine

- **parameter**: 8 bit number in Acc
- **result**: 8 bit result in Acc

```
double: mov r7,a    ; copy number to R7
        add a,r7 ; add it back to Acc
        ret ; return with value in Acc
```

```
main mov r0,#$50    ; value in M[50]
        lcall double
```

```
double: mov a,@r0    ; get the number
        add a,@r0 ; add it to itself
        mov @r0,a ; overwrite original number
        ret
```

```
main mov r0,#$50 ; value in M[50]
        lcall double
```

---

---
Multi-parameter subroutine example

eg: copy($20, $60, 8); // copy 8 bytes from M[20] to M[60]

procedure copy (src, dst :^BYTE; count:BYTE);
begin
  repeat
    dst^ := src^;
    inc(src);
    inc(dst);
    dec(count)
  until count = 0;
end;

; COPY a block of memory
; Parameters
;   Acc : number of bytes to copy
;   r0 : source – address of block to copy
;   r1 : destination address

  copy: jz done ; exit if finished
  mov  r7,a  ; save count in R7
  lup: mov  a, @r0  ; get a byte
        mov @r1,a ; and save it
        inc r0  ; point at next pair
        inc r1  ; of bytes
        djnz r7,lup ; if any more, again
  done: ret

; To call copy – setup parameters, then call subroutine
; The order the parameters are loaded is irrelevant.

  mov  r0, #20  ; copy 8 bytes from
  mov  r1, #60  ; M[20] to M[60]
  mov  a, 8
  lcall copy