
ELEC2041 Microprocessors and Interfacing

Lectures 13: Making Decisions in C/Assembly Language - I

<http://webct.edtec.unsw.edu.au/>

March, 2006

Saeid Nooshabadi

saeid@unsw.edu.au

ELEC2041 lec13-decision-1.1

Saeid Nooshabadi

Overview

- C/Assembly if, goto, if-else
- C /Assembly Loops: goto, while
- Test for less Than, Greater Than, etc
- C/Assembly case/switch statement
- Conclusion

ELEC2041 lec13-decision-1.2

Saeid Nooshabadi

Review (#1/2)

- **Big idea in CS&E; compilation** to translate from one level of abstraction to lower level
 - Generally single HLL statement produces many assembly instructions
 - Also hides address calculations (byte vs. word addressing)
- **Design of an Assembly Language like ARM shaped by**
 - 1) Desire to keep hardware simple:
e.g., most operations have 3 operands
 - 2) Smaller is faster:
e.g., ARM has 16 registers

ELEC2041 lec13-decision-1.3

Saeid Nooshabadi

Review (#2/2)

- **ARM assembly language thus far:**
 - Instructions: add, sub, mov, orr, and, bic, eor, mul,
 - ldr, str At most one assembly instruction per line
 - Comments start with; to end of line
 - Operands: registers
 - r0 - r3 → a1 - a4
(correspond to C functions arguments. Used for scratch pad too!)
 - r4 - r10 → v1 - v7
(correspond to function variables)
 - Operands: memory
 - Memory[0], Memory[4], Memory[8],
... , Memory[4294967292]

ELEC2041 lec13-decision-1.4

Saeid Nooshabadi

Baby Quiz

- What are the three different ways to obtain a data operand you have seen?
- **Immediate** - data is IN the instruction
 - add r1, r1, #24
- **Register Direct** - data is IN a register
 - the register number is in the instruction
 - add r1, r1, r2
- **Base plus offset** - data is IN memory
 - the register number is in the instruction
 - the base address is in the register
 - the offset is in the instruction/ offset index register number in instruction
 - ldr r1, [r2, #24] / ldr r1, [r2,r3]
 - These are called **Addressing Modes**

ELEC2041 lec13-decision-1.5

Saeid Nooshabadi

C Decisions (Control Flow): if statements

- 2 kinds of if statements in C
 - if (*condition*) *statement*
 - if (*condition*) *statement1* else *statement2*
- Following code is same as 2nd if

```
if (condition) goto L1;
    statement2;
    goto L2;
L1: statement1;
L2:
```

 - Not as elegant as if-else, but same meaning

ELEC2041 lec13-decision-1.6

Saeid Nooshabadi

ARM decision instructions (control flow) (#1/2)

- Decision instruction in ARM:
 - cmp register1, register2 ;compare register1 with ; register2
 - beq L1 ; branch to L1 if equalis “Branch if (registers are) equal”
Same meaning as C:
 - if (register1==register2) goto L1
- Complementary ARM decision instruction
 - cmp register1, register2
 - bne L1• bne is “Branch if (registers are) not equal”
Same meaning as C:
 - if (register1!=register2) goto L1
- Called **conditional branches**

ELEC2041 lec13-decision-1.7

Saeid Nooshabadi

ARM decision instructions (control flow) (#2/2)

- Decision instruction in ARM:
 - cmp register1, #immediate ;compare register1 with ; immediate number
 - beq L1 ; branch to L1 if equalis “Branch if (register and immediate are) equal”
Same meaning as C:
 - if (register1==immediate) goto L1
- Complementary ARM decision instruction
 - cmp register1, #immediate
 - bne L1• bne is “Branch if (register and immediate are) not equal”
Same meaning as C:
 - if (register1!=immediate) goto L1
- Called **conditional branches**

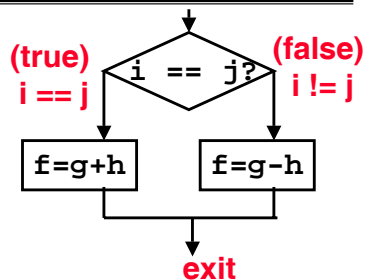
ELEC2041 lec13-decision-1.8

Saeid Nooshabadi

Compiling C if into ARM Assembly

Compile by hand

```
if (i == j) f=g+h;
else f=g-h;
Mapping f: v1, g: v2,
h: v3, i: v4, j: v5
```



Start with branch:

```
cmp v4, v5
beq L-true           ; branch to L-True
                    ; if i==j
```

Follow with false part

```
sub v1,v2,v3        ; f=g-h
```

Compiling C if into ARM Assembly

Need instruction that always transfers control to skip over true part

- ARM has branch: `b label ; goto "label"`

```
sub v1,v2,v3        ; f=g-h
b L-true
```

Next is true part

```
L-true: add v1,v2,v3 ; f=g+h
```

Followed by exit branch label

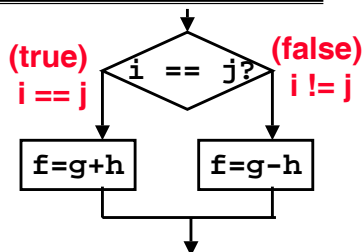
```
L-true:
L-true:
L-true:
```

Compiling C if into ARM: Summary

Compile by hand

C `if (i == j) f=g+h;`
`else f=g-h;`

Mapping f: v1, g: v2,
h: v3, i: v4, j: v5



A `cmp v4,v5`
R `beq L-true` ; branch i==j
M `sub v1,v2,v3` ; (false)
`b L-true` ; go to Exit
`L-true: add v1,v2,v3` ;(true)
`L-true:`

Note: Compiler supplies labels for branches not found in HLL code; often it flips the condition to branch to false part

Motoring with Microprocessors

- Thanks to the magic of microprocessors and embedded systems, our cars are becoming safer, more efficient, and entertaining.
- The average middle-class household includes over 40 embedded processors. About half are in the garage. Cars make a great vehicle for deploying embedded processors in huge numbers. These processors provide a ready source of power, ventilation, and mounting space and sell in terrific quantities.
- How many embedded processors does your car have?
- If you've got a late-model luxury sedan, two or three processors might be obvious in the GPS navigation system or the automatic distance control. Yet you'd still be off by a factor of 25 or 50. The current 7-Series BMW and S-class Mercedes boast about 100 processors apiece. A relatively low-profile Volvo still has 50 to 60 baby processors on board. **Even a boring low-cost econobox has a few dozen different microprocessors in it.**
- Your transportation appliance probably has more chips than your Internet appliance.
- New cars now frequently carry 200 pounds of electronics and more than a mile of wiring.

ELEC2041 Reading Materials (Week #5)

- Week #5: Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6. We use chapters 3 and 5
- ARM Architecture Reference Manual –On CD ROM
- A copy of the article by Cohen, D. “On holy wars and a plea for peace (data transmission).” *Computer*, vol.14, (no.10), Oct. 1981. p.48-54, is place on the class website.

Loops in C/Assembly

◦ Simple loop in C

```
Loop:  g = g + A[i];
       i = i + j;
       if (i != h) goto Loop;
```

◦ (g,h,i,j:v2,v3,v4,v5; base of A[]:v6):

◦ 1st fetch A[i]

```
Loop: ldr a1,[v6, v4, lsl #2]
       ;(v6+v4*4)=addr A[i]
       ;a1=A[i]
```

Simple Loop (cont)

◦ Add value of A[i] to g and then j to i

```
g = g + a1
i = i + j;
```

◦ (g,h,i,j:v2,v3,v4,v5):

```
add v2,v2,a1      ; g = g+A[i]
add v4,v4,v5      ; i = i + j
```

The final instruction branches back to Loop if i != h:

```
cmp v4,v3
bne Loop          ; goto Loop
                  ; if i!=h
```

Loops in C/Assembly: Summary

```
Loop:  g = g + A[i];
C      i = i + j;
       if (i != h) goto Loop;
```

◦ (g,h,i,j:v2,v3,v4,v5; base of A[]:v6):

```
Loop: ldr a1,[v6, v4, lsl #2]
       ;(v6+v4*4)=addr A[i]
       ;a1=A[i]
A      add v2,v2,a1      ; g = g+A[i]
R      add v4,v4,v5      ; i = i + j
M      cmp v4,v3
       bne Loop          ; goto Loop
       ; if i!=h
```

while in C/Assembly:

◦ Although legal C, almost never write loops with `if, goto`: use `while`, or `for` loops

◦ Syntax: `while(condition) statement`

```
while (save[i] == k)
    i = i + j;
```

◦ 1st load `save[i]` into a scratch register (`i, j, k`: `v4, v5, v6`: base of `save[]:v7`):

```
Loop:  ldr  a1, [v7, v4, lsl #2]
        ;v7+v4*4=addr of save[i]
        ;a1=save[i]
```

While in C/Assembly (cont)

◦ Loop test: exit if `save[i] != k`
(`i, j, k`: `v4, v5, v6`: base of `save[]:v7`)

```
cmp  a1, v6
bne  Exit      ;goto Exit
                ;if save[i]!=k
```

◦ The next instruction adds `j` to `i`:

```
add  v4, v4, v5 ; i = i + j
```

◦ End of loop branches back to the `while` test at top of loop. Add the `Exit` label after:

```
      b      Loop      ; goto Loop
Exit:
```

While in C/Assembly: Summary

```
C      while (save[i]==k)
        i = i + j;
```

(`i, j, k`: `v4, v5, v6`: base of `save[]:v7`)

```
Loop:  ldr  a1, [v7, v4, lsl #2]
        ;v7+v4*4=addr of save[i]
        ;a1=save[i]
```

```
A      cmp  a1, v6
R      bne  Exit      ;goto Exit
M      ;if save[i]!=k
        add  v4, v4, v5 ; i = i + j
        b    Loop      ; goto Loop
```

```
Exit:
```

Beyond equality tests in ARM Assembly (#1/2)

◦ So far `==, !=`, What about `<` or `>`?

- `cmp register1, register2`

- `blt L1`

is “Branch if (register1 < register2)”

Same meaning as C:

- `if (register1 < register2) go to L1`

◦ Complementary ARM decision instruction

- `cmp register1, register2`

- `bge L1`

• `bge` is “Branch if (register1 >= register2)”

Same meaning as C:

- `if (register1 >= register2) go to L1`

Beyond equality tests in ARM Assembly (#2/2)

◦ Also

• `cmp register1, #immediate`

• `blt L1`

is “Branch if (register1 <#immediate)”

Same meaning as C:

• `if (register1 < immediate) go to L1`

◦ Complementary ARM decision instruction

• `cmp register1, #immediate`

• `bge L1`

• `bge` is “Branch if (register1 >= #immediate)”

Same meaning as C:

`if (register1 >= immediate) go to L1`

If less_than in C/Assembly

```
C if (g < h) { ... }
```

```
      cmp v1, v2           ; v1 < v2 (g < h)
      blt Less            ; if (g < h)
R     b    noLess        ; if (g >= h)
      Less:
      ...
M     noLess:
```

Alternative Code (Generated by a good compiler)

```
      cmp v1, v2           ; v1 < v2 (g < h)
      bge noLess        ; if (g >= h)
      ...
      noLess:
      b    Less          ; if (g < h)
```

Some Branch Conditions

◦ `b` Unconditional

◦ `bal` Branch Always

◦ `beq` Branch Equal

◦ `bne` Branch Not Equal

◦ `blt` Branch Less Than

◦ `ble` Branch Less Than or Equal

◦ `bgt` Branch Greater Than

◦ `bge` Branch Greater Than or Equal

◦ Full Table Page 64 Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6.

What about unsigned numbers?

◦ Conditional branch instructions `blt`, `ble`, `bgt`, etc, assume signed operands (defined as `int` in C). The equivalent instructions for unsigned operands (defined as `unsigned` in C), are:

◦ `blo` Branch Lower (unsigned)

◦ `bls` Branch Less or Same (unsigned)

◦ `bhi` Branch Higher (unsigned)

◦ `bhs` Branch Higher or Same (Unsigned)

◦ `v1 = FFFF FFFAhex`, `v2 = 0000 FFFAhex`

◦ What is result of

```
cmp v1, v2
```

```
bgt L1
```

```
cmp v1, v2
```

```
bhi L1
```

Signed VS Unsigned Comparison

- $v1 = \text{FFFF FFFA}_{\text{hex}}$, $v2 = \text{0000 FFFA}_{\text{hex}}$
- $v1 < v2$ (signed interpretation)
- $v1 > v2$ (unsigned interpretation)
- What is result of

cmp v1, v2

bgt L1

...

L1:

Branch NOT taken

cmp v1, v2

bhi L1

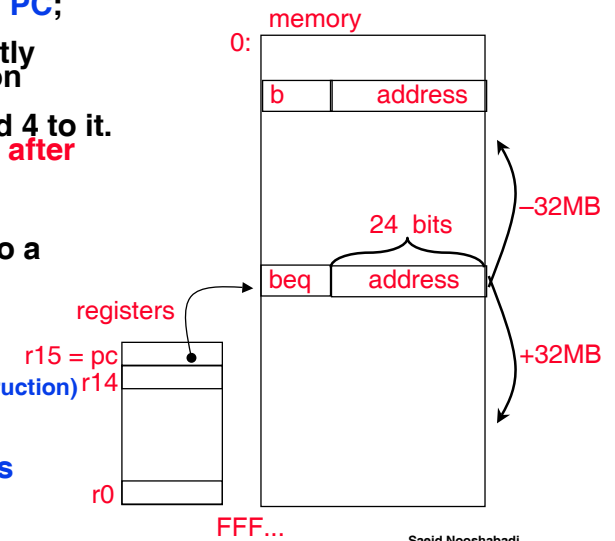
...

L1:

Branch Taken

Branches: PC-relative addressing

- Recall register **r15** in the machine also called **PC**;
- points to the currently executing instruction
- Most instructions add 4 to it. (pc increments by 4 after execution of most instructions)
- Branch changes it to a specific value
- Branch adds to it
 - 24-bit signed value (contained in the instruction)
 - Shifted left by 2 bits
- Labels => addresses



C case/switch statement

- Choose among four alternatives depending on whether **k** has the value 0, 1, 2, or 3

```
switch (k) {
    case 0: f=i+j; break; /* k=0*/
    case 1: f=g+h; break; /* k=1*/
    case 2: f=g-h; break; /* k=2*/
    case 3: f=i-j; break; /* k=3*/
}
```

Case/switch via chained if-else, C

- Could be done like chain of if-else

```
if(k==0) f=i+j;
else if(k==1) f=g+h;
else if(k==2) f=g-h;
else if(k==3) f=i-j;
```

Case/switch via chained if-else, C/Asm.

- Could be done like chain of if-else

```
if(k==0) f=i+j;
else if(k==1) f=g+h;
else if(k==2) f=g-h;
else if(k==3) f=i-j;
(f, i, j, g, h, k: v1, v2, v3, v4, v5, v6)
```

C

A

R

M

```

cmp    v6, #0
      bne    L1      ; branch k!=0
add    v1, v2, v3   ; k=0 so f=i+j
      b     Exit    ; end of case
L1:   cmp    v6, #1
      bne    L2      ; branch k!=1
add    v1, v4, v5   ; k=1 so f=g+h
      b     Exit    ; end of case
L2:   cmp    v6, #2
      bne    L3      ; branch k!=2
      sub    v1, v4, v5 ; k=2 so f=g-h
      b     Exit    ; end of case
L3:   cmp    v6, #3
      bne    Exit    ; branch k!=2
      sub    v1, v2, v3 ; k=3 so f=i-j
Exit:

```

ELEC2041 lec13-decision-1.29

Saeid Nooshabadi

Case/Switch via Jump Address Table

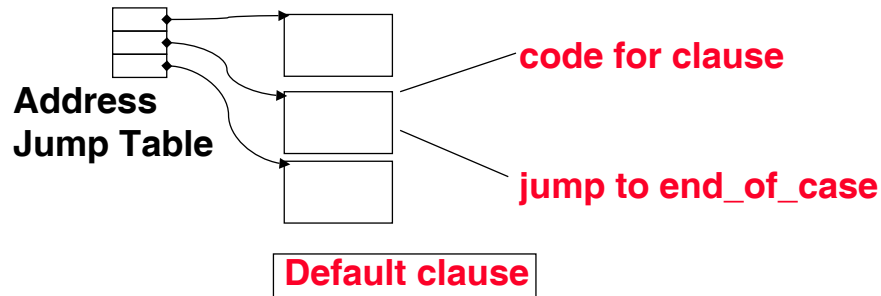
- Notice that last case must wait for $n-1$ tests before executing, making it slow
- Alternative tries to go to all cases equally fast: **jump address table**
 - Idea: encode alternatives as a table of addresses of the cases
 - Table an array of words with addresses corresponding to case labels
 - Program indexes into table and jumps
- ARM instruction “**ldr pc, []**” unconditionally branches to address L1 (Changes PC to address of L1)

ELEC2041 lec13-decision-1.30

Saeid Nooshabadi

Idea for Case using Jump Table

- check within range
- get address of target clause from target array
- jump to target address



end_of_case:

ELEC2041 lec13-decision-1.31

Saeid Nooshabadi

Case/Switch via Jump Address Table (#1/3)

- Use k to index a jump address table, and then jump via the value loaded
- 1st test that k matches 1 of cases ($0 \leq k \leq 3$); if not, the code exits

($k: v6, v7$: Base address of `JumpTable[k]`)

```

cmp    v6, #0      ;Test if k < 0
      blt    Exit    ;if k<0,goto Exit
cmp    v6, #3      ;Test if k > 3
      bgt    Exit    ;if k>3,goto Exit

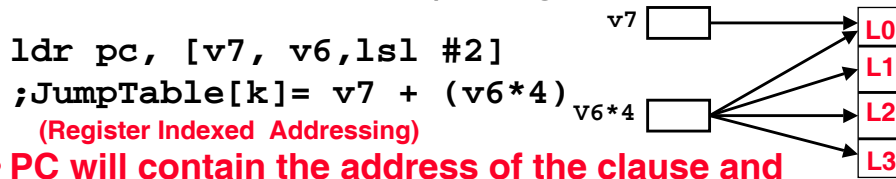
```

ELEC2041 lec13-decision-1.32

Saeid Nooshabadi

Case/Switch via Jump Address Table (#2/3)

- Assume 4 sequential words (4 bytes) in memory, with base address in $v7$, have addresses corresponding to labels L0, L1, L2, L3.
- Now use $(4*k)$ ($k:v6$) to index table of words and load the clause address from the table (Address of labels L0, L1, L2, L3) to register pc .



(Register Indexed Addressing)

- PC will contain the address of the clause and execution starts from there.

Case/Switch via Jump Address Table (#3/3)

- Cases jumped to by `ldr pc, [v7, v6, lsl #2]`:

```
L0: add v1,v2,v3      ; k=1 so f=i+j
    b   Exit          ; end of case
L1: add v1,v4,v5      ; k=1 so f=g+h
    b   Exit          ; end of case
L2: sub v1,v4,v5      ; k=2 so f=g-h
    b   Exit          ; end of case
L3: sub v1,v2,v3      ; k=3 so f=i-j
```

Exit:

Question: How do I define a jumtable and put the Case addresses in?

Answer: Easy

JumpTable:

```
.word L0,L1, L2, L3
```

Question: How do I put the base address of the JumpTable in register $v7$?

Answer: Easy

Will discuss later!

Jump Address Table: Summary

$(k, f, i, j, g, h$ Base address of `JumpTable[k]`)
 $:v6, v1, v2, v3, v4, v5, v7)$

```
cmp v6, #0           ;Test if k < 0
blt Exit             ;if k<0,goto Exit
cmp v6, #3           ;Test if k >3
bgt Exit             ;if k>3,goto Exit
```

```
ldr pc, [v7, v6, lsl #2]
;JumpTable[k]= v7 + (v6*4)
```

```
L0: add v1,v2,v3      ; k=1 so f=i+j
    b   Exit          ; end of case
L1: add v1,v4,v5      ; k=1 so f=g+h
    b   Exit          ; end of case
L2: sub v1,v4,v5      ; k=2 so f=g-h
    b   Exit          ; end of case
L3: sub v1,v2,v3      ; k=3 so f=i-j
```

Exit:

More example on Jump Tables on CD-ROM

If there is time, do it yourself:

- Compile this C code into ARM:

```
sum = 0;
for (i=0; i<10; i=i+1)
    sum = sum + A[i];
```

- $sum:v1, i:v2, \text{base address of } A:v3$

(If time allows) Do it yourself:

C `sum = 0;`
`for (i=0;i<10;i=i+1)`
`sum = sum + A[i];`

• `sum:v1, i:v2, base address of A:v3`

A
R
M `mov v1, #0`
`mov v2, #0`
`Loop: ldr a1,[v3,v2,ls1 #2] ; a1=A[i]`
`add v1, v1, a1 ; sum = sum+A [i]`
`add v2, v2, #1 ; increment i`
`cmp v2, #10 ; Check(i<10)`
`bne Loop ; goto loop`

“And in Conclusion ...” (#1/2)

◦ HLL decisions (if, case) and loops (while, for) use same assembly instructions

- Comparison: `cmp` in ARM
- Conditional branches: `beq, bne` in ARM
- Unconditional Jumps: `b, ldr pc,____, mov pc, ____` in ARM
- Case/Switch: either chained if-else or jump table + `ldr pc, ____`

“And in Conclusion...” (#1/2)

◦ New Instructions:

`cmp`
`beq`
`bne`
`bgt`
`bge`
`blt`
`ble`
`bhi`
`bhs`
`blo`
`bls`