Fundamentals of Computer Architecture

ARM – Conditional Execution, Branches

Teacher: Lobar Asretdinova

Conditional Execution

Example:

```
CMP   X5, X9          ; performs X5-X9
                      ; sets condition flags

SUBEQ X1, X2, X3      ; executes if X5==X9 (Z=1)
ORRMI X4, X0, X9      ; executes if X5-X9 is
                      ; negative (N=1)
```

Conditional Execution

Example:

```
CMP    X5, X9          ; performs X5-X9
                       ; sets condition flags

SUBEQ X1, X2, X3       ; executes if X5==X9 (Z=1)
ORRMI X4, X0, X9       ; executes if X5-X9 is
                       ; negative (N=1)
```

Suppose X5 = 17, X9 = 23:

`CMP` performs: 17 – 23 = -6 (Sets flags: *N*=1, *Z*=0, *C*=0, *V*=0)

`SUBEQ` **doesn't execute** (they aren't equal: *Z*=0)

`ORRMI` **executes** because the result was negative (*N*=1)

Programming Building Blocks

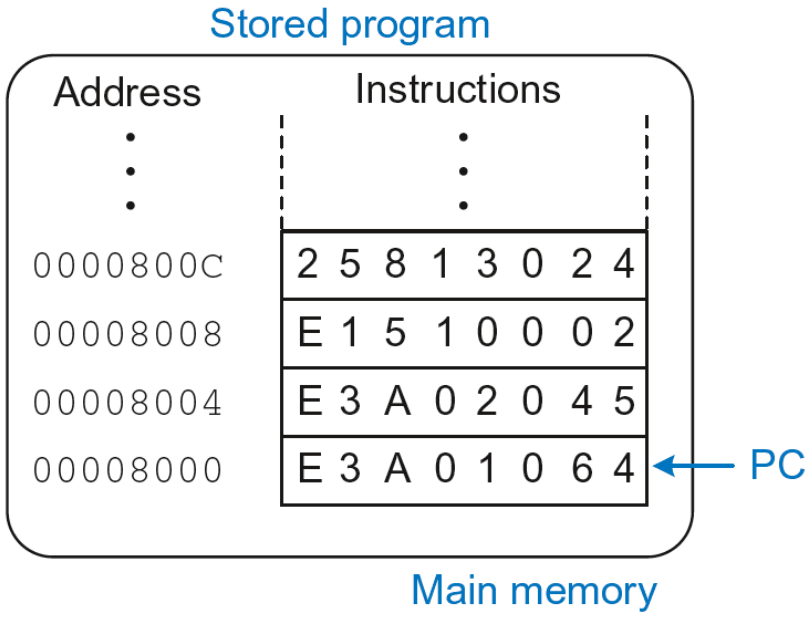
- Data-processing Instructions
- Conditional Execution
- Branches
- High-level Constructs:
  - if/else statements
  - for loops
  - while loops
  - arrays
  - function calls

Branching

- Branches enable out of sequence instruction execution
- Types of branches:
  - **Branch (`B`)**
    - branches to another instruction
  - **Branch and link (`BL`)**
    - discussed later
- Both can be conditional or unconditional

The Stored Program

Assembly code	Machine code
`MOV R1, #100`	`0xE3A01064`
`MOV R2, #69`	`0xE3A02045`
`CMP R1, R2`	`0xE1510002`
`STRHS R3, [R1, #0x24]`	`0x25813024`



Unconditional Branching (B)

ARM assembly

```
   MOV X2, #17          ; X2 = 17
   B   TARGET           ; branch to target
   ORR X1, X1, #0x4     ; not executed

TARGET:
      SUB X1, X1, #78       ; X1 = X1 + 78
```

Unconditional Branching (B)

ARM assembly

```
    MOV X2, #17          ; X2 = 17
    B   TARGET           ; branch to target
    ORR X1, X1, #0x4     ; not executed

TARGET
        SUB X1, X1, #78      ; X1 = X1 + 78
```

Labels (like TARGET) indicate instruction location.

Labels can't be reserved words (like ADD, ORR, etc.)

The Branch Not Taken

ARM Assembly

```
  MOV   X0, #4          ; X0 = 4
  ADD   X1, X0, X0      ; X1 = X0+X0 = 8
  CMP   X0, X1          ; sets flags with X0-X1
  BEQ   THERE           ; branch not taken (Z=0)
  ORR   X1, X1, #1      ; X1 = X1 OR X1 = 9
THERE
  ADD X1, X1, 78        ; R1 = R1 + 78 = 87
```

Compiling Loop Statements

- C code:

```
while (save[i] == k) i += 1;
```

  - i in x22, k in x24, address of save in x25

- Compiled ARMv8 code:

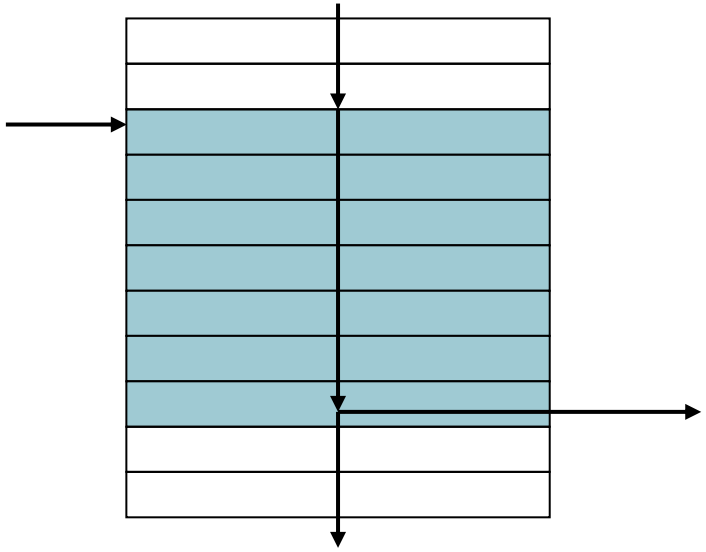
```
Loop: LSL    X10,X22,#3
      ADD    X10,X10,X25
      LDUR   X9,[X10,#0]
      SUB    X11,X9,X24
      CBNZ   X11,Exit
      ADDI   X22,X22,#1
      B      Loop
Exit: …
```

X10 = i·2^3 ⇒ 8 (byte for address)

Compare Branch Not Zero

Basic Blocks

- A basic block is a sequence of instructions with
  - No embedded branches (except at end)
  - No branch targets (except at beginning)



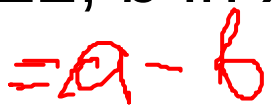
- A compiler identifies basic blocks for optimization
- An advanced processor can accelerate execution of basic blocks

More Conditional Operations

- Condition codes, set from arithmetic instruction with S-suffix (ADDS, ADDIS, ANDS, ANDIS, SUBS, SUBIS)
  - negative (N): result had 1 in MSB
  - zero (Z): result was 0
  - overlow (V): result overflowed
  - carry (C): result had carryout from MSB
- Use subtract to set flags, then conditionally branch:
  - **B.EQ**
  - **B.NE**
  - **B.LT** (less than, signed), **B.LO** (less than, unsigned)
  - **B.LE** (less than or equal, signed), **B.LS** (less than or equal, unsigned)
  - **B.GT** (greater than, signed), **B.HI** (greater than, unsigned)
  - **B.GE** (greater than or equal, signed),
  - **B.HS** (greater than or equal, unsigned)

Conditional Example

- if (a > b) a += 1;
  - a in X22, b in X23



```
SUBS X9,X22,X23  // use subtract to make comparison
B.LTE Exit       // conditional branch
ADDI X22,X22,#1
Exit:
```

Signed vs. Unsigned

- Signed comparison
- Unsigned comparison
- Example
  - X22 = 1111 1111 1111 1111 1111 1111 1111 1111
  - X23 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `X22 < X23 # signed`
    - –1 < +1
  - `X22 > X23 # unsigned`
    - +4,294,967,295 > +1

Programming Building Blocks

- Data-processing Instructions
- Conditional Execution
- Branches
- **High-level Constructs:**
  - **if/else statements**
  - **for loops**
  - **while loops**
  - arrays
  - function calls

if Statement

C Code

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



f = f - i;
```

if Statement

C Code

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

f = f - i;
```

ARM Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

  CMP X3, X4        ; set flags with X3-X4
  BNE L1            ; if i!=j, skip if block
  ADD X0, X1, X2    ; f = g + h

L1:
  SUB X0, X0, X2    ; f = f - i
```

if Statement

C Code

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

f = f - i;
```

ARM Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

  CMP X3, X4        ; set flags with R3-R4
  BNE L1            ; if i!=j, skip if block
  ADD X0, X1, X2    ; f = g + h

L1
  SUB X0, X0, X2    ; f = f - i
```

Assembly tests opposite case (`i != j`) of high-level code (`i == j`)

if Statement: Alternate Code

C Code

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

ARM Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

 CMP   X3, X4       ; set flags with R3-R4
 ADDEQ X0, X1, X2   ; if (i==j) f = g + h
 SUB   X0, X0, X2   ; f = f - i
```

if Statement: Alternate Code

Original

```
 CMP R3, R4
 BNE L1
 ADD R0, R1, R2
L1:
 SUB R0, R0, R2
```

Alternate Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

 CMP   X3, X4       ; set flags with R3-R4
 ADDEQ X0, X1, X2   ; if (i==j) f = g + h
 SUB   X0, X0, X2   ; f = f - i
```

if Statement: Alternate Code

Original

```
 CMP X3, X4
 BNE L1
 ADD X0, X1, X2
L1
 SUB X0, X0, X2
```

Alternate Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

 CMP   X3, X4       ; set flags with X3-X4
 ADDEQ X0, X1, X2   ; if (i==j) f = g + h
 SUB   X0, X0, X2   ; f = f - i
```

Useful for **short** conditional blocks of code

if/else Statement

C Code **ARM Assembly Code**

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


else
  f = f - i;
```

if/else Statement

C Code

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

else
  f = f - i;
```

ARM Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

  CMP X3, X4        ; set flags with X3-X4
  BNE L1            ; if i!=j, skip if block
  ADD X0, X1, X2    ; f = g + h
  B   L2            ; branch past else block
L1
  SUB X0, X0, X2    ; f = f - i
L2
```

if/else Statement: Alternate Code

C Code

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

ARM Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

CMP   X3, X4       ; set flags with X3-X4
ADDEQ X0, X1, X2   ; if (i==j) f = g + h
SUBNE X0, X0, X2   ; else f = f - i
```

if/else Statement: Alternate Code

Original

```
 CMP X3, X4
 BNE L1
 ADD X0, X1, X2
 B   L2
L1
 SUB X0, X0, X2
L2
```

Alternate Assembly Code

```
;X0=f, X1=g, X2=h, X3=i, X4=j

 CMP   X3, X4       ; set flags with X3-X4
 ADDEQ X0, X1, X2   ; if (i==j) f = g + h

 SUBNE X0, X0, X2   ; else f = f - i
```

while Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

ARM Assembly Code

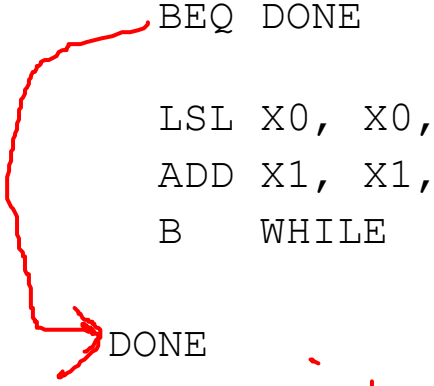
while Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

ARM Assembly Code

```
; R0 = pow, X1 = x
  MOV   X0, #1          ; pow = 1
  MOV   X1, #0          ; x = 0

WHILE
  CMP X0, #128          ; X0-128
  BEQ DONE              ; if (pow==128)
                        ; exit loop
  LSL X0, X0, #1        ; pow=pow*2
  ADD X1, X1, #1        ; x=x+1
  B   WHILE             ; repeat loop


DONE
```



while Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

ARM Assembly Code

```
; X0 = pow, X1 = x
  MOV   X0, #1          ; pow = 1
  MOV   X1, #0          ; x = 0

WHILE
  CMP X0, #128          ; X0-128
  BEQ DONE              ; if (pow==128)
                        ; exit loop
  LSL X0, X0, #1        ; pow=pow*2
  ADD X1, X1, #1        ; x=x+1
  B   WHILE             ; repeat loop

DONE
```

Assembly tests for the opposite case (pow == 128) of the C code (pow != 128).

for Loops

```
for (initialization; condition; loop operation)
   statement
```

- `initialization:` executes before the loop begins
- `condition:` is tested at the beginning of each iteration
- `loop operation:` executes at the end of each iteration
- `statement:` executes each time the condition is met

for Loops

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

ARM Assembly Code

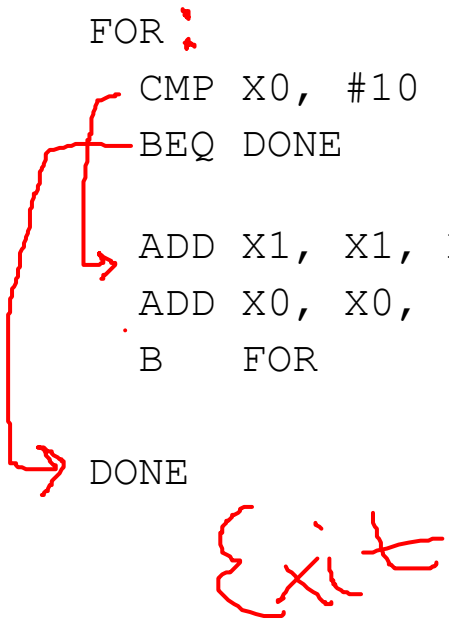
for Loops

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

ARM Assembly Code

```
; X0 = i, X1 = sum
  MOV   X0, #1          ; i = 1
  MOV   X1, #0          ; sum = 0

FOR:
  CMP X0, #10           ; X0-10
  BEQ DONE              ; if (i==10)
                        ; exit loop
  ADD X1, X1, X0        ; sum=sum + i
  ADD X0, X0, #1        ; i = i + 1
  B   FOR               ; repeat loop


DONE
```



for Loops: Decremented Loops

In ARM, decremented loop variables are more efficient

for Loops: Decremented Loops

In ARM, decremented loop variables are more efficient

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=9; i!=0; i=i-1)
   sum = sum + i;
```

ARM Assembly Code

```
; X0 = i, X1 = sum
  MOV   X0, #9          ; i = 9
  MOV   X1, #0          ; sum = 0

FOR :
  ADD   X1, X1, X0      ; sum=sum + i
  SUBS  X0, X0, #1      ; i = i - 1
                        ; and set flags
  BNE   FOR             ; if (i!=0)
                        ; repeat loop
```

for Loops: Decremented Loops

In ARM, decremented loop variables are more efficient

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=9; i!=0; i=i-1)
   sum = sum + i;
```

ARM Assembly Code

```
; X0 = i, X1 = sum
  MOV   X0, #9          ; i = 9
  MOV   X1, #0          ; sum = 0

FOR
  ADD   X1, X1, X0      ; sum=sum + i
  SUBS  X0, X0, #1      ; i = i - 1
                        ; and set flags
  BNE   FOR             ; if (i!=0)
                        ; repeat loop
```

Saves 2 instructions per iteration:

- Decrement loop variable & compare: `SUBS X0, X0, #1`
- Only 1 branch – instead of 2