CPE 221: Computer Organization

04 ARM Conditions, Branching and Loops

rahul.bhadani@uah.edu

Rahul Bhadani

Announcement

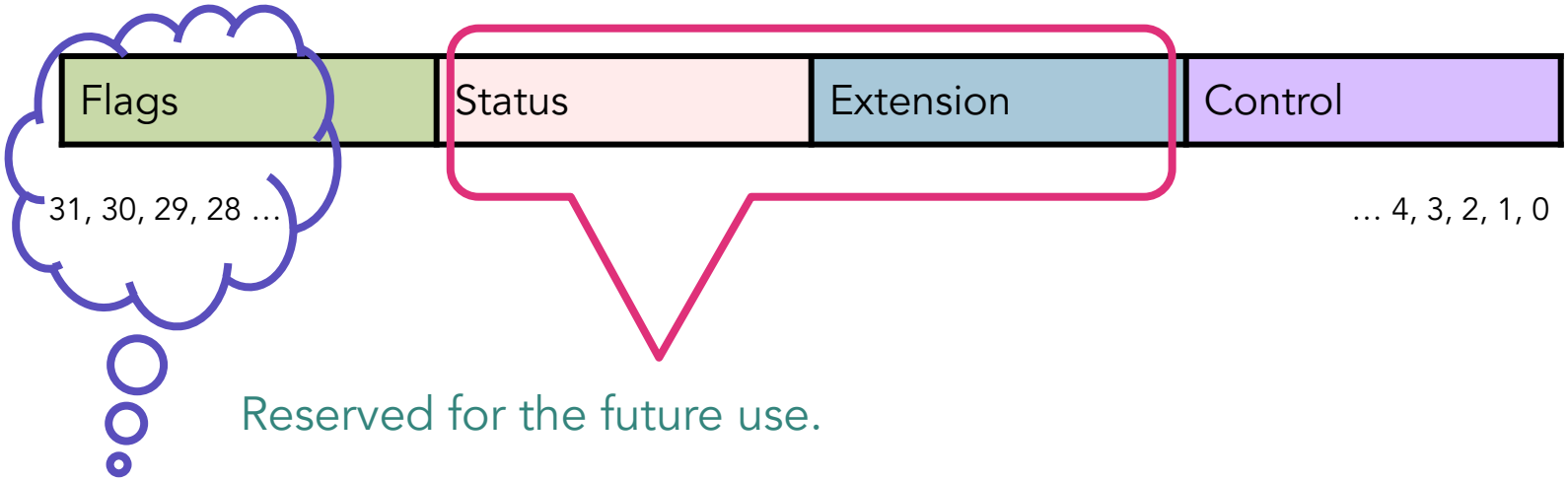
Homework 03 Due: Feb 14, 2025, 11:59 PM

Exam 1: February 19, 2025, Wednesday: 4:20 PM - 5:40 PM

- 1 Page, One-sided , Handwritten Sheet Allowed
- Calculator Allowed
- Close Notes, Close Books
- May ask to write ARM Programming
- Convert C Code to ARM Code
- Will cover topics: Number Systems, 2s complements Floating Points, Digital Logic, Register-Transfer Language, Transistor/Gate Basics, K-map Minimization, Truth Tables, ARM Instructions, Questions on Disassembly and Memory Map of ARM Programs

Conditional Flags

Current Program Status Register (CPSR)



N Z C V Q IT[1:0] J

Together, they are called Application Program Status Register (APSR)

- N: Instruction result is negative, i.e. bit 31 of the result is 1.
- Z: Instruction result is zero. bit 30
- C: Instruction results in a carry out. bit 29
- V: Instruction causes an overflow. bit 28
- Q: Cumulative saturation/Sticky Flag. bit 27
- IT[1:0]: Reserved. bits 26, 25
- J: Reserved. bit 24. Some newer version indicates whether the core is in Jazelle state (we will discuss this later).

Flag Examples

`SUB R1, R2, R`

- If result < 0 → `N=1`
- If result = 0 → Z=1

`ADDS R0, R1, R2`

- Instructions with an `S` suffix (e.g., ADDS, SUBS) update the CPSR flags
- Unsigned overflow → `C=1`
- Signed overflow → `V=1`

Condition Mnemonics

Some common suffixes for conditional execution:

- `EQ` (`Z=1`) Equal
- `NE` (`Z=0`) Not Equal
- `MI` (`N=1`) Minus
- `PL` (`N=0`) Plus
- `CS/HS` (`C=1`) Carry Set Or Unsigned Higher or Same
- `CC/LO` (`C=0`) Carry Clear / Unsigned Lower
- `VS` (`V=1`) Overflow Set
- `VC` (`V=0`) Overflow Clear
- `GT`, `LT`, `GE`, `LE` (signed comparisons)

Conditional Execution

```
CMP R0, R1      @ Compare R0 and R1 which sets the flags
MOVGT R2, #100  @ Move if R0 > R1
ADDLE R2, #10   @ Add if R0 ≤ R1
```

Branching

Branching Alters Sequential Program Execution

→ PC is incremented by 4 bytes ***after*** each instruction is executed to point to the next instruction. (Why 4 bytes?)

→ Branch instruction changes the PC to point to an address where execution should jump to.

Simple Branch

`B`: For Simple Branching

→ Can also be used with condition mnemonics.

→ Useful for implementing if/else

Example: Nested If/Else

High-level Code

```
if (a > b) {
    if (b > c) {
        d = 1;
    } else { // else2
        d = 2;
    }
} else { //else1
    d = 3;
}
```

ARM Assembly Code

```
   LDR R0, =a          @ Load address of a into R0
   LDR R1, =b          @ Load address of b into R1
   LDR R2, =c          @ Load address of c into R2
   LDR R0, [R0]        @ Load value of a into R0
   LDR R1, [R1]        @ Load value of b into R1
   LDR R2, [R2]        @ Load value of c into R2
   CMP R0, R1          @ Compare a and b
   BLE else1           @ If a <= b, jump to else1
   CMP R1, R2          @ Compare b and c
   BLE else2           @ If b <= c, jump to else2
   MOV R3, #1          @ d = 1
   B end               @ Jump to end
else2:
   MOV R3, #2          @ d = 2
   B end               @ Jump to end
else1:
   MOV R3, #3          @ d = 3
end:
   LDR R4, =d          @ Load address of d into R4
   STR R3, [R4]        @ Store the result in d
```

Example: Nested If/Else with Logical Conditions

High-level Code

```
if (a > b && b > c) {
    d = 1;
} else {
    d = 0;
}
```

ARM Assembly Code

```
    LDR R0, =a          @ Load address of a into R0
    LDR R1, =b          @ Load address of b into R1
    LDR R2, =c          @ Load address of c into R2
    LDR R0, [R0]        @ Load value of a into R0
    LDR R1, [R1]        @ Load value of b into R1
    LDR R2, [R2]        @ Load value of c into R2
    CMP R0, R1          @ Compare a and b
    BLE else            @ If a <= b, jump to else
    CMP R1, R2          @ Compare b and c
    BLE else            @ If b <= c, jump to else
    MOV R3, #1          @ d = 1
    B end               @ Jump to end
else:
    MOV R3, #0          @ d = 0
end:
    LDR R4, =d          @ Load address of d into R4
    STR R3, [R4]        @ Store the result in d
```

Example: For Loop: Array Summation

ARM Assembly Code

```
    LDR R0, =array      @ Load address of array into R0
    MOV R1, #0          @ Initialize sum (R1 = 0)
    MOV R2, #0          @ Initialize counter (R2 = 0)
    MOV R3, #5          @ Set number of items to add (R3 = 5)

for_loop:
    CMP R2, R3          @ Compare R2 with R3
    BGE end_for         @ If R2 >= R3, exit loop
    LDR R4, [R0], #4    @ Load array element into R4 and increment pointer
    ADD R1, R1, R4      @ Add R4 to sum (R1 = R1 + R4)
    ADD R2, R2, #1      @ Increment counter (R2 = R2 + 1)
    B for_loop          @ Repeat loop

end_for:

done: B done

array:
    .word 1, 2, 3, 4, 5 @ Define array
```

Example: while Loop with String Length Calculation

ARM Assembly Code

```
    LDR R0, =string     @ Load address of string into R0
    MOV R1, #0          @ Initialize length counter (R1 = 0)

while_loop:
    LDRB R2, [R0], #1   @ Load byte from string into R2 and increment pointer
    CMP R2, #0          @ Compare R2 with null terminator
    BEQ end_while       @ If R2 == 0, exit loop
    ADD R1, R1, #1      @ Increment length counter (R1 = R1 + 1)
    B while_loop        @ Repeat loop

end_while:
    @ Loop ends here
done: B done

string:
    .asciz "Elephant"      @ Define null-terminated string
```