ENGN2219/COMP6719
Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

Australian
National
University

Plan: Week 5

Last week: Finite State Machines, Timing, Pipelining

This Week: Finish looking at memories

This Week: Instruction Set Architecture (ISA)

Conditional Execution

- We may or may not want to execute a specific instruction
  - Example: *Only execute an instruction if a specific condition is satisfied*
- ARM allows conditional execution of instructions
  - **Step # 1**: Instruction sets the condition flags (negative, zero, carry, overflow)
  - **Step # 2**: Subsequent instructions execute based on the state of the condition flags
- Condition (status) flags are set by ALU
  - They are contained in a register called the Current Program Status Register (CPSR)

Setting the Condition Flags

- **Method 1:** Use the compare instruction

```
CMP   R5,     R6
```

- The instruction subtracts the second source operand from the first operand (R6 – R5)
- The instruction does not save any result
- Set flags as follows
  - Is 0, Z = 1
  - Is negative, N = 1
  - Causes a carry out, C = 1
  - Causes a signed overflow, V = 1

Setting the Condition Flags

- **Method 2:** Append the instruction mnemonic with **S**

```
ADDS  R1,    R2,    R3
```

- The instruction adds source operands R2 and R3
- It sets the flags (**S**)
- It saves the result in R1

Condition Mnemonics

- Instruction may be conditionally executed based on condition flags
- Condition for execution is encoded as a ***condition mnemonic*** appended to the instruction mnemonic

```
CMP     R1,   R2
SUBNE   R3,   R5,   R8
ADDEQ   R1,   R2,   R3
```

- **NE** and **EQ** are condition mnemonics
- SUB executes only if R1 is not equal to R2 (i.e., Z = 0)

Condition Mnemonics

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same	C
0011	CC / LO	Carry clear / Unsigned lower	$\bar{C}$
0100	MI	Minus / Negative	N
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set	V
0111	VC	No overflow / Overflow clear	$\bar{V}$
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z\ OR\ \bar{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	$Z\ OR\ (N \oplus V)$
1110	AL (or none)	Always / unconditional	ignored

Instructions that affect condition flags

Type	Instructions	Condition Flags
Add	ADDS, ADCS	*N*, *Z*, *C*, *V*
Subtract	SUBS, SBCS, RSBS, RSCS	*N*, *Z*, *C*, *V*
Compare	CMP, CMN	*N*, *Z*, *C*, *V*
Shifts	ASRS, LSLS, LSRS, RORS, RRXS	*N*, *Z*, *C*
Logical	ANDS, ORRS, EORS, BICS	*N*, *Z*, *C*
Test	TEQ, TST	*N*, *Z*, *C*
Move	MOVS, MVNS	*N*, *Z*, *C*
Multiply	MULS, MLAS, SMLALS, SMULLS, UMLALS, UMULLS	*N*, *Z*

Example – 1

- R5 = 17 and R9 = 23
- Will the SUBEQ and ORRMI instructions execute?
  - NZCV = ????

```
CMP     R5,   R9
SUBEQ   R1,   R2,   R3
ORRMI   R4,   R0,   R9
```

Example – 2

- R2 = 0x80000000 and R3 = 0x00000001
  - NZCV = ????
  - Which instructions will execute?

```
CMP     R2,   R3
ADDEQ   R4,   R5,   #78
ANDHS   R7,   R8,   R9
ORRMI   R10,  R11,  R12
EORLT   R12,  R7,   R10
```

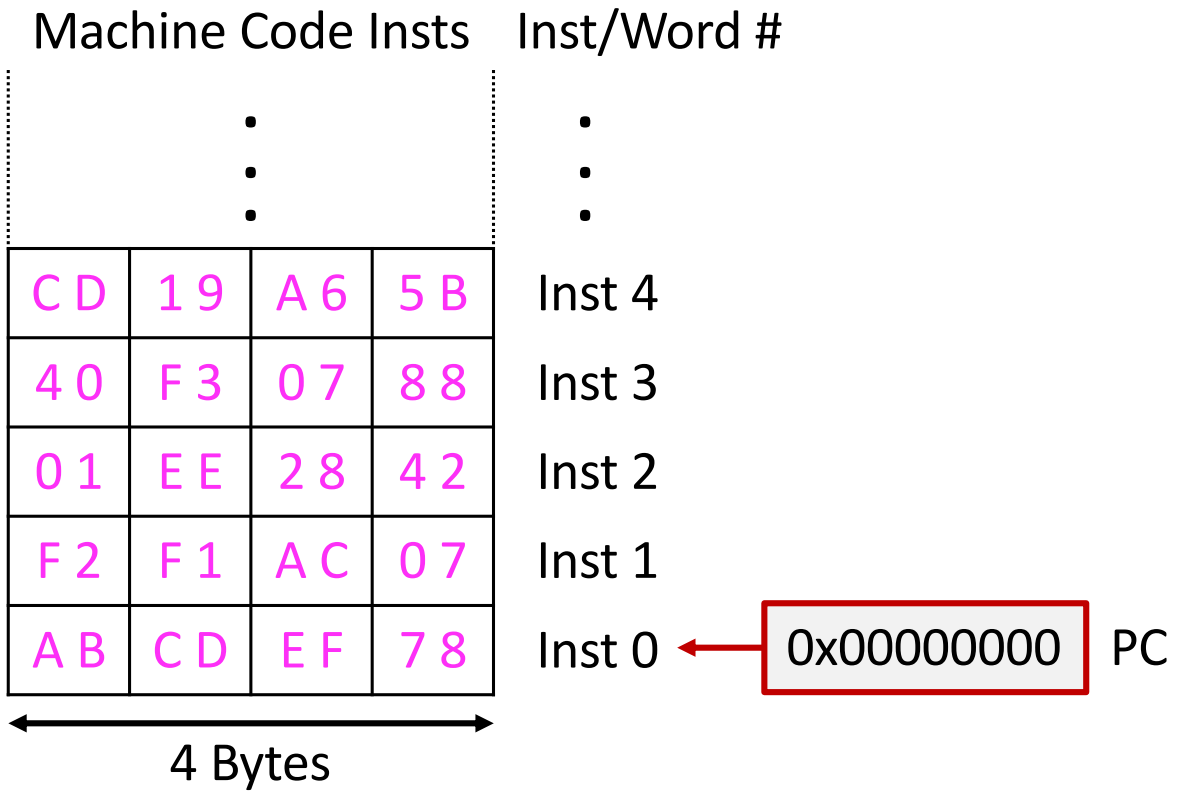
Branch Instructions

- Decision making is an important advantage of computers
  - Perform a different task based on the input
  - if and if/else statements
  - for and while loops
  - switch/case statements
- ARM provides branch instructions to skip/repeat code
- Branch instructions are more efficient/powerful than conditional execution
  - E.g., conditional execution cannot implement loops

Program Counter (PC)

- PC is a special register that points to the current instruction
- To execute the next instruction, the CPU increments the PC by 4

- In ARM, the register R15 is the PC
- Without any branch instruction, the CPU increments the PC and executes the next instruction



Branch Instructions

- Branch instructions change the program flow or the flow of code execution
  - Normal execution (without branch): Increment PC by 4 bytes (PC = PC + 4) to fetch the next instruction
  - Branch instruction: Change PC to the address of the target instruction, and fetch the target instruction
- Target instruction
  - Target instruction may be few or many bytes away
  - Remember: Target is an address in main memory

Type of Branches

- Branch (B)
  - Branches to another instruction
- Branch and Link (BL)
  - A special branch instruction to provide support for functions in high-level languages
- The branch instructions can be conditional or unconditional
- Examples of conditional branch instructions
  - B**EQ**
  - B**NE**

Unconditional Branch

- Branch is *unconditional* and thus *always taken*
- After encountering the branch, the CPU executes SUB instead of ORR

```
        ADD   R1,   R2   #17
        B     TARGET
        ORR   R1,   R1,  R3
        AND   R3,   R1,  #0xFF
TARGET
        SUB   R1,   R1,  #78
```

- The label TARGET is translated by the assembler into a memory address

Conditional Branch

Branch instructions can execute conditionally based on the condition mnemonics

- When the code reaches BEQ, the Z condition flag is 0 because R0 is not equal to R1
- Branch is *not taken*
- The next instruction executed is the ORR instruction

```
        MOV     R0,     #4
        ADD     R1,     R0,     R0
        CMP     R0,     R1
        BEQ     THERE
        ORR     R1,     R1,     R1
THERE
        ADD     R1,     R1,     #78
```

if Statement

C code:

```
if (apples == oranges)
    f = i + 1;
f = f – i;
```

- The assembly code below checks for the opposite condition in C code
- Skips the if block if the condition is not satisfied
- If the branch is *not taken,* if block is executed

ARM Assembly Code

```
        CMP     R0,     R1          apples == oranges?
        BNE     L1                  if not equal, skip if block
        ADD     R2,     R3      #1  if block: f = i + 1
L1
        SUB     R2,     R2,     R3  f = f – i
```

R0 = apples, R1 = oranges, R2 = f, R3 = i

if Statement

- The assembly code tests the opposite condition of the one in C code
- Skips the if block if the condition is **not** satisfied
- If the branch is *not taken,* if block is executed

- Question: Is there a different way of writing the assembly code for the if statement?
  - Using the BEQ instruction instead of BNE (Try it yourself)
  - Using conditional execution (next)

if Statement with Conditional Execution

C code:

```
if (apples == oranges)
        f = i + 1;
f = f – i;
```

ARM Assembly Code

```
        CMP     R0,     R1
        ADDEQ   R2,     R3      #1
        SUB     R2,     R2,     R3
```

apples == oranges?

if block: f = i + 1 on equality (Z = 1)

f = f – i

R0 = apples, R1 = oranges, R2 = f, R3 = i

if Statement with Conditional Execution

- One fewer instruction
  - *The solution is shorter and faster*
  - *Branch instructions sometimes introduce extra delay*
- As the if block becomes longer, the branch becomes valuable
  - Conditional execution requires needless fetching of instructions
  - Tedious to write code

if/else Statement

- Execute one of two blocks of code depending on the condition
- When the condition in the if block is met, the if block is executed
- Otherwise, the else block is executed

if/else Statement

C code:

```
if (apples == oranges)
    f = i + 1;
else
    f = f – i;
```

ARM Assembly Code

```
    CMP   R0,   R1
    BNE   L1
    ADD   R2,   R3   #1
    B     L2
L1
    SUB   R2,   R2,  R3
L2
```

if not equal, skip if block

skip else block

R0 = apples, R1 = oranges, R2 = f, R3 = i

if/else Statement

C code:

```
if (apples == oranges)
        f = i + 1;
else
        f = f – i;
```

Homework Exercise: Find an alternative way to write the if/else statement.

if/else Statement with Conditional Execution

C code:

```
if (apples == oranges)
        f = i + 1;
else
        f = f – i;
```

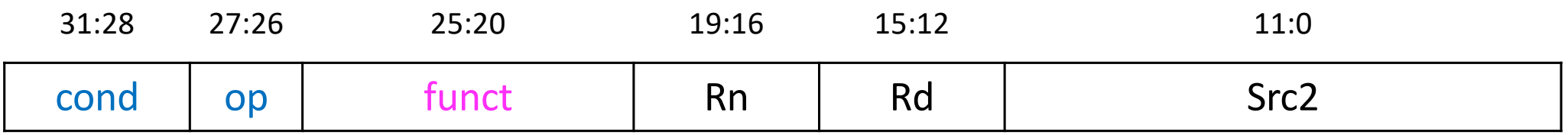
ARM Assembly Code

```
        CMP     R0,     R1
        ADDEQ   R2,     R3      #1
        SUBNE   R2,     R2,     R3
```

Machine Language

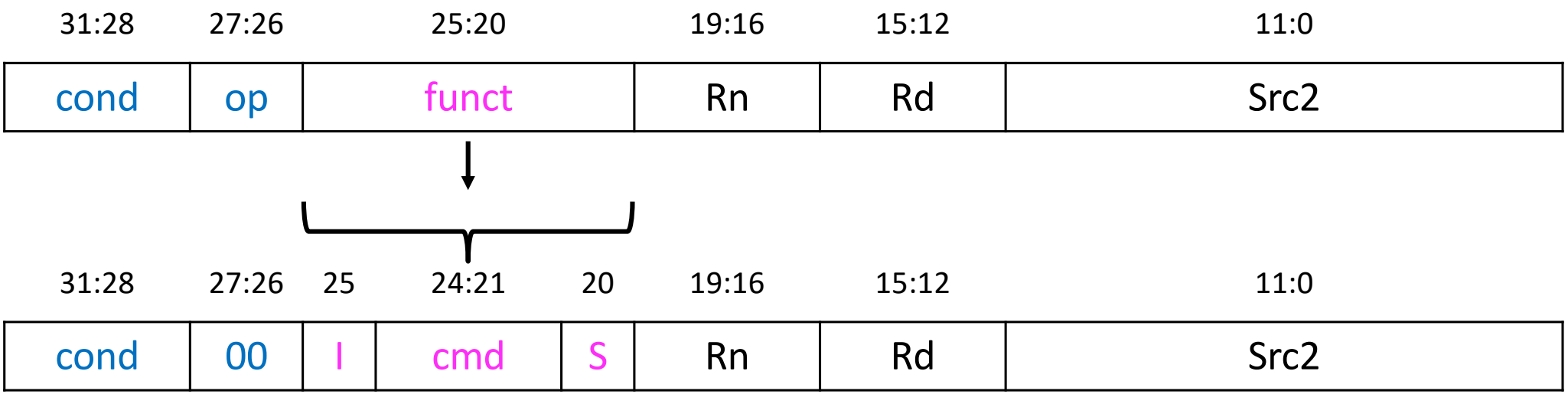
- Instructions are stored in memory as 32-bit words
- ARM defines three instruction formats
  - Data processing (DP)
  - Memory
  - Branch
- Instruction *fields* encode the instruction operation and its operands
- Binary encoding of three instruction formats will allow us to build a CPU for the ARM ISA

Instruction Format – 1: DP



- Operands
  - Rn: first source operand register (0000, 0001, ..., 1111)
  - Src2: second source register or immediate
  - Rd: destination register
- Control fields
  - cond: specifies conditional execution (1110 for unconditional)
  - op: the operation code or opcode (00)
  - funct: the function/operation to perform

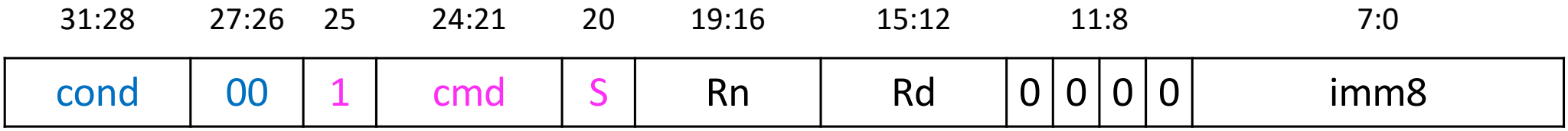
Instruction Format – 1: DP



- op = 00 for DP instructions
- cmd specifies the specific DP instruction (0100 for ADD and 0010 for SUB)
- I-bit
  - I = 0: Src2 is a register
  - I = 1: Src2 is an immediate
- S-bit: 1 if sets condition flags

Two DP Formats (Src2 Variations)

Immediate (assume 11:8 are 0 for now)



Register (assume 11:4 are zero for now)