Lancaster University

SCC.131: ARM Assembly serialisation

Plan of the day

Lancaster University

- As assembly master you should know by now:
  - How to perform numerical operations with registers and constants
  - How to change the execution flow of a program

- You must be wondering by now

  “Computers use bytes and I write in ASCII”

- Convert ARM Assembly to binary.
- Deep dive in the ISA design.

The Instruction

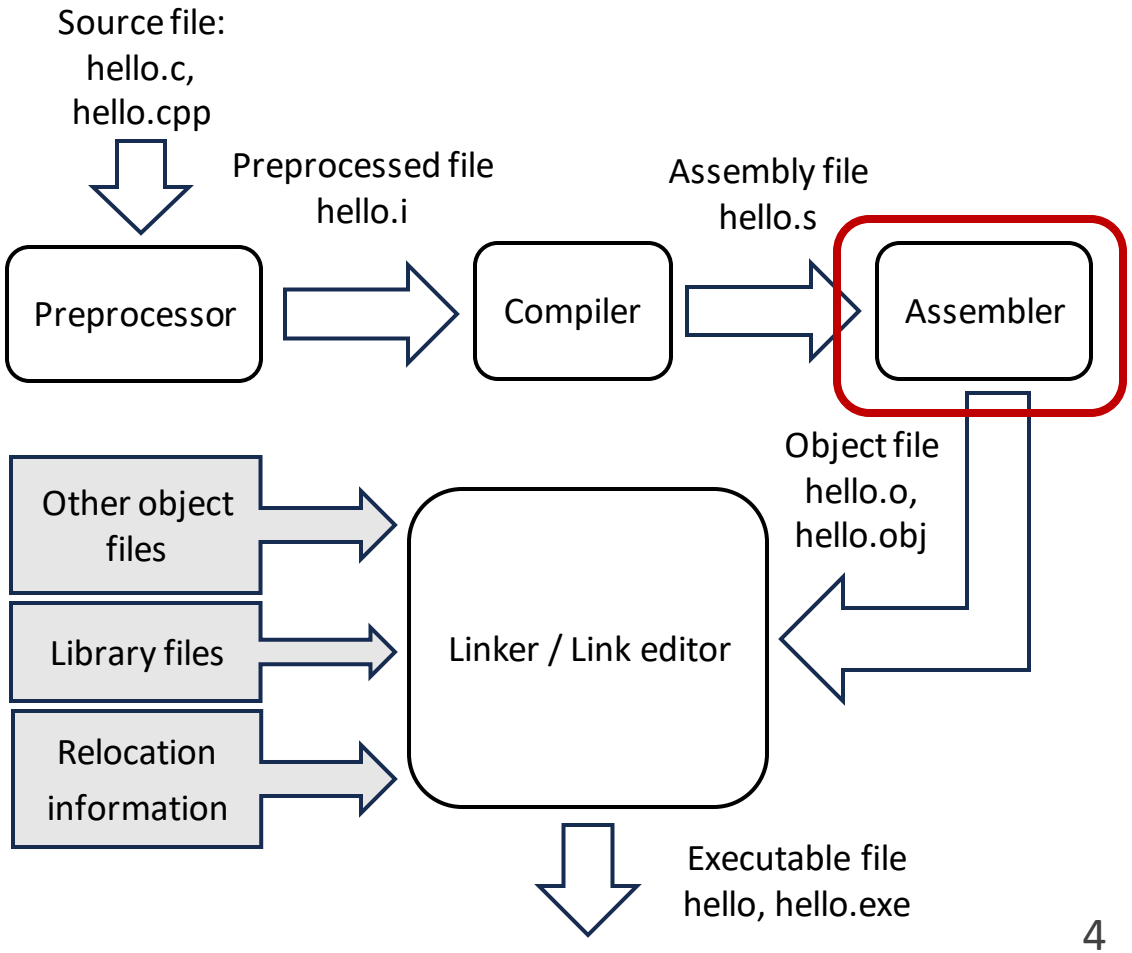
Lancaster University

- An instruction is the most basic unit of computer processing
  - Instructions are words in the language of a computer
  - Instruction Set Architecture (ISA) is the vocabulary
- The language of the computer can be written as
  - Machine language: Computer-readable representation (that is, 0's and1's)
  - Assembly language: Human-readable representation
- We will study ARM (in detail)
- Principles are similar in all ISAs (x86, SPARC, RISC-V, …)

Creation of executable file

Lancaster University

1. Preprocessor
   - Macros, `#include` directives, #xxxx statements.
   - Output: “pure” C code.
2. Compiler
   - Transforms C into assembly code.
   - Not machine code: still human-readable.
   - Dependent on machine architecture.
3. Assembler
   - Creates machine code, stored in an object file.
4. Linker
   - Combines several object files together.



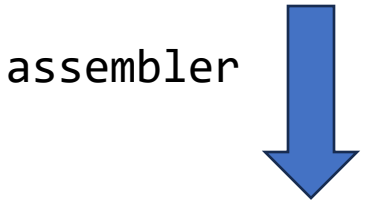
Assembler

Lancaster University

- The assembler convert each instruction into a 16/32-bit representation.
- Well-defined translation process.
- Each binary instruction contains:
  - Instruction
  - Registers
  - Immediate values
- ARM-Cortex M3 supports ARM T32 and Thumb-2 (thumb by default).

```
add r0, r1, r2
add r1, r2
```

assembly



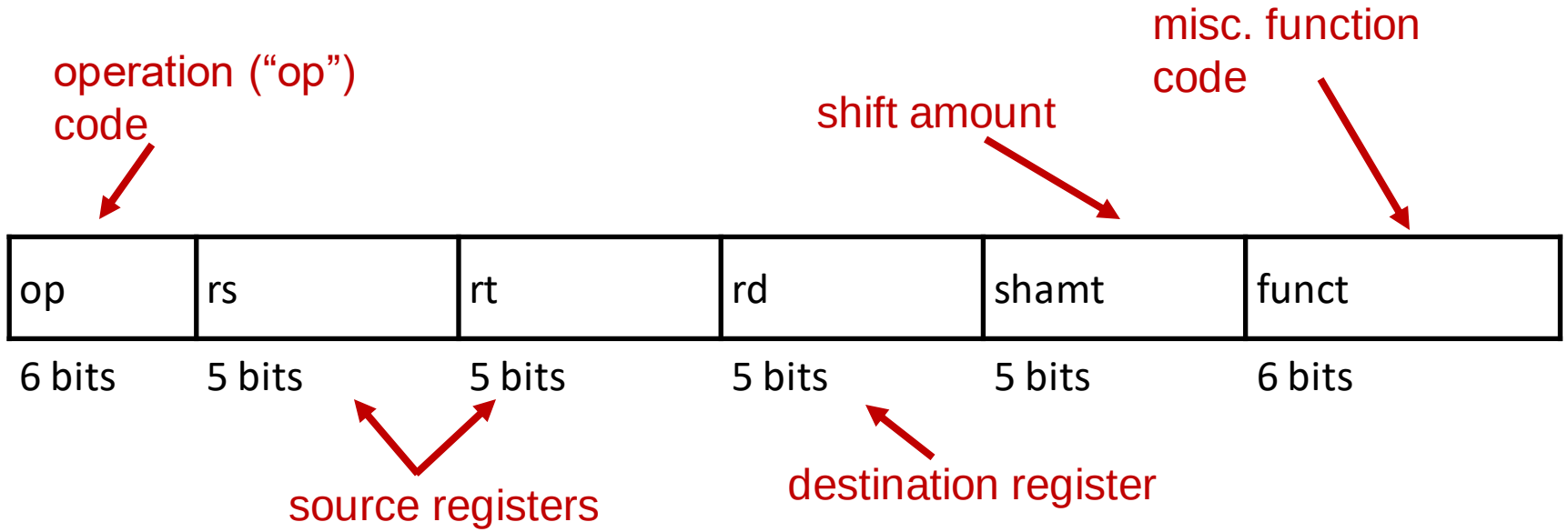
E	B	0	1	0	0	0	2
4	4	0	2	X	X	X	X

Machine code

MIPS machine code (not ARM) (1)

Lancaster University

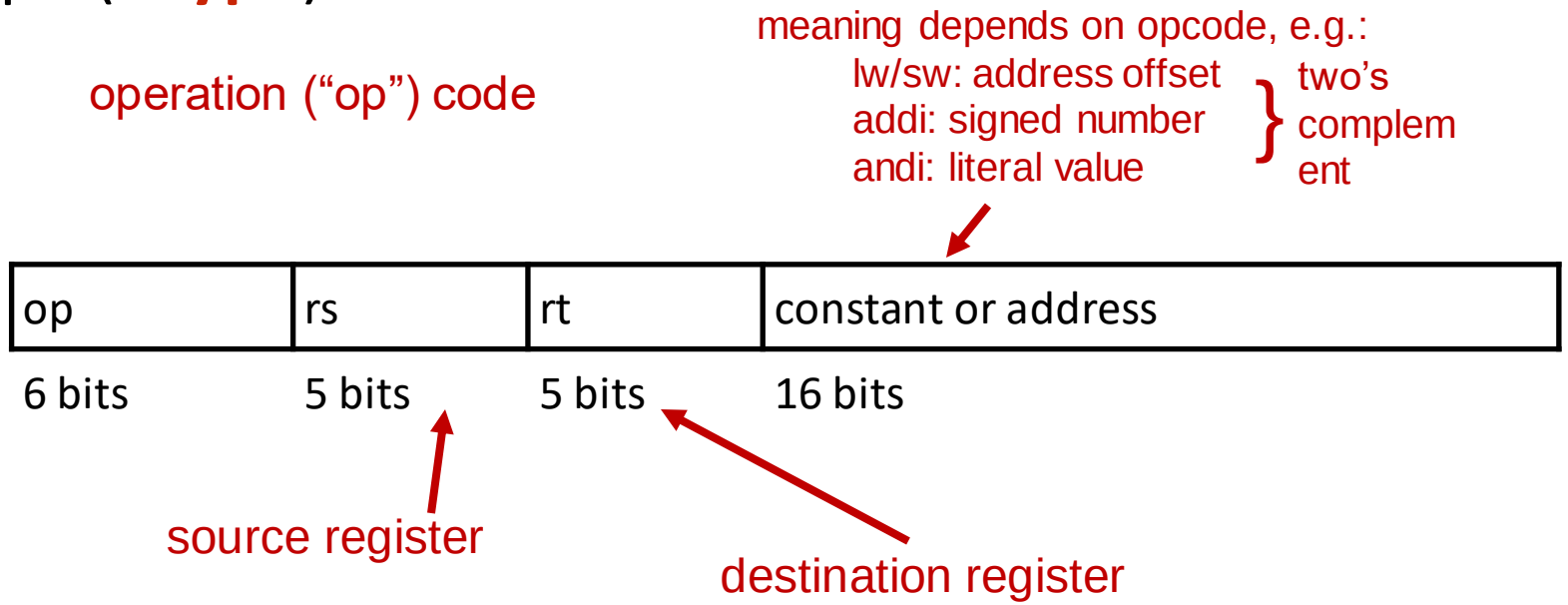
- 3 basic types of MIPS instructions
  - **Register type (R-type)**
  - Immediate type (**I-type**)
  - Jump type (**J-type**)



MIPS machine code (not ARM) (2)

Lancaster University

- 3 basic types of MIPS instructions
  - Register type (R-type)
  - **Immediate type (I-type)**
  - Jump type (**J-type**)



Example Instruction – Add ARM

ARM instruction

```
add r0, r1, r2
```

ARM instruction type

```
ADD{S}<c>.W <Rd>, <Rn>, <Rm>{, <shift>}
```

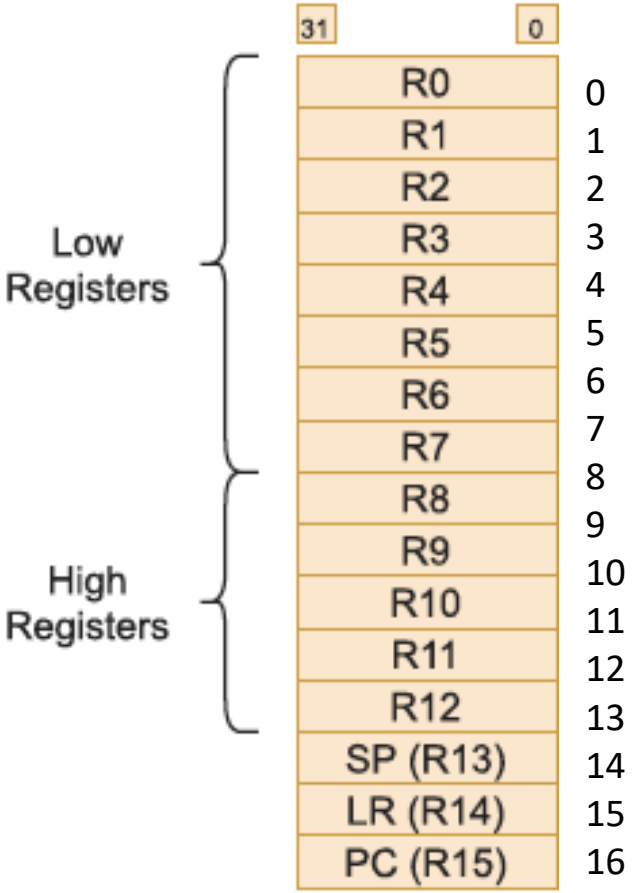
ARM specifications

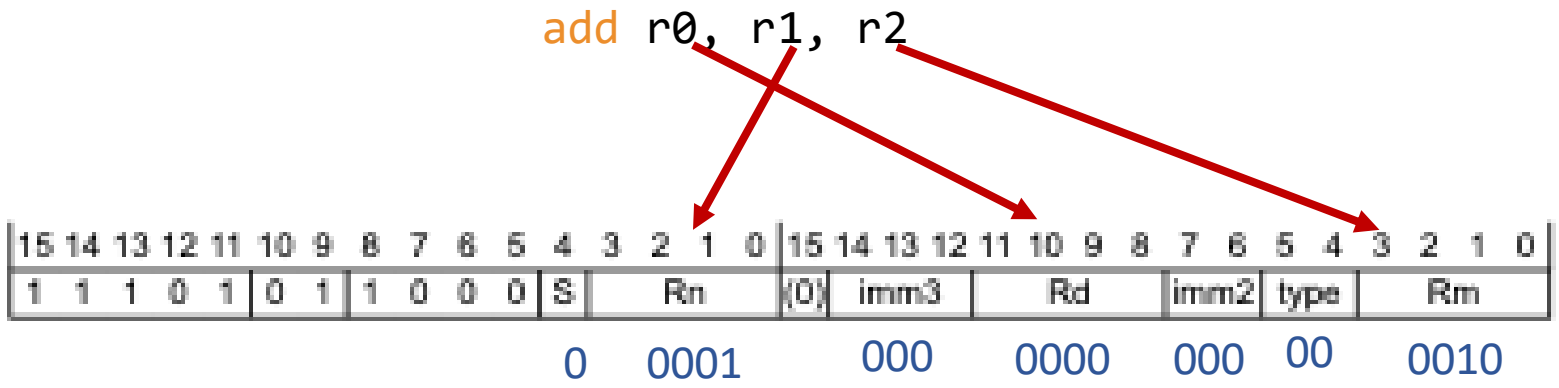
Instruction opcode
(unique for each instruction)

Registers

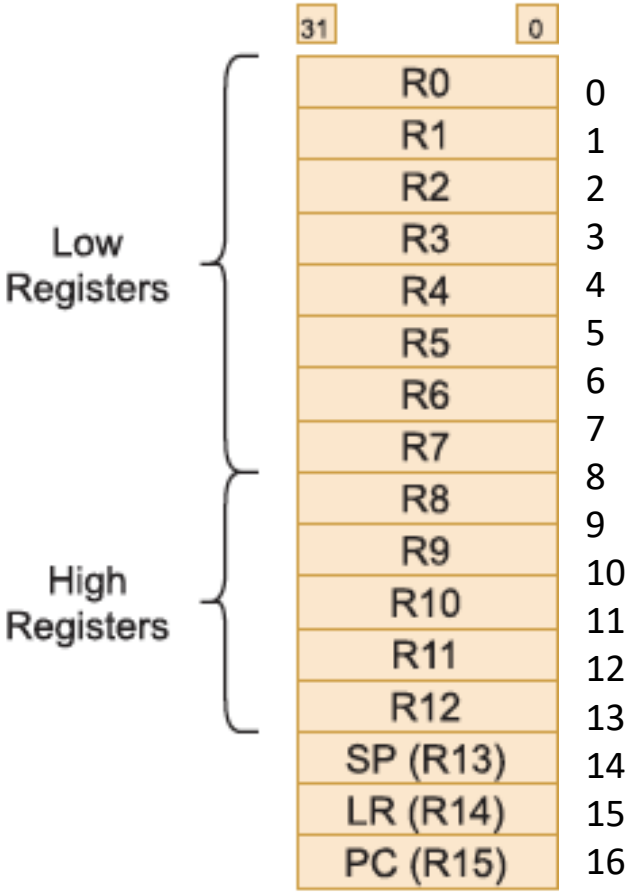
Optional Shift

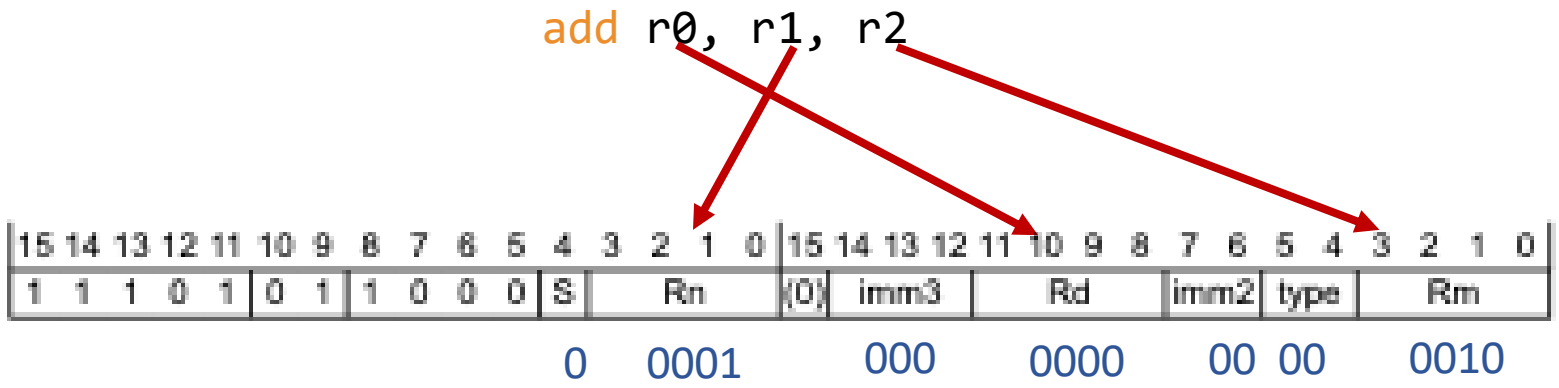
Example Instruction – Add ARM





Example Instruction – Add ARM





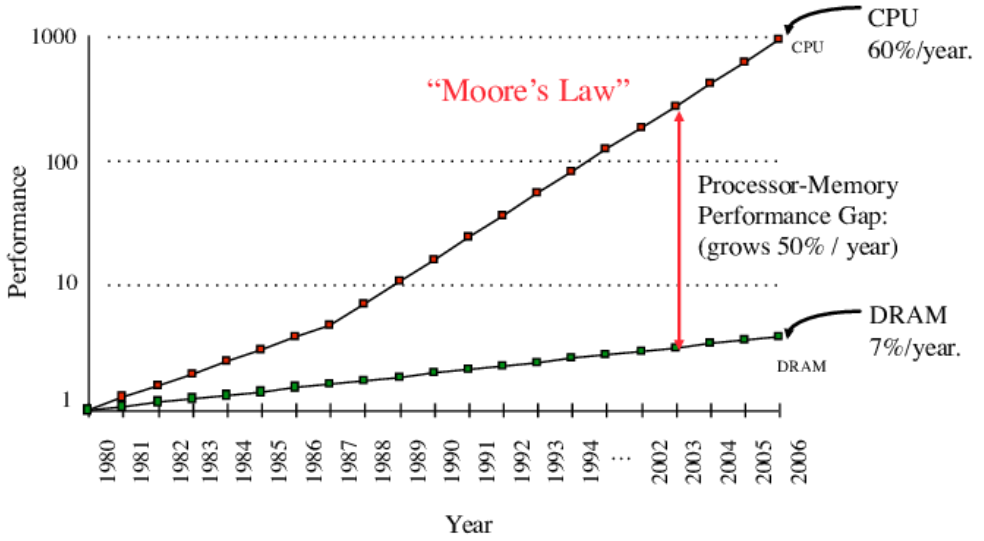
1110 1011 0000 0000 1000 0000 0000 0000 0010

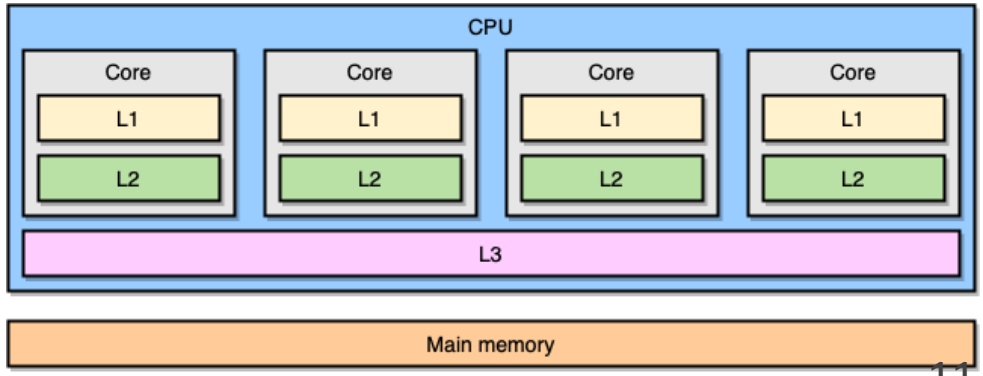
E B 0 0 8 0 0 0 2

Moore's Law revisited

Lancaster University

- It is easier to speed up the CPU than the memory.
  - Off-chip memory (SRAM) is slower than on-chip memory (M1,M2 cache).
  - Flash is slower to read or write than RAM.
  - Fast RAM is more expensive than slow RAM.
- Design for high-performance CPUs (e.g. x86)
  - Use caches (small fast RAM) to make main memory (large slow RAM, flash) look faster at a low cost.
- Design for low-performance CPUs (e.g. ARM-Cortex)
  - Put memory on-chip with CPU. RAM, flash ROM
  - Increase flash ROM bandwidth by widening the memory bus, adding prefetch buffer, branch target buffer, etc.
  - Add cache
  - **Change instruction set size to reduce instruction bandwidth needed**





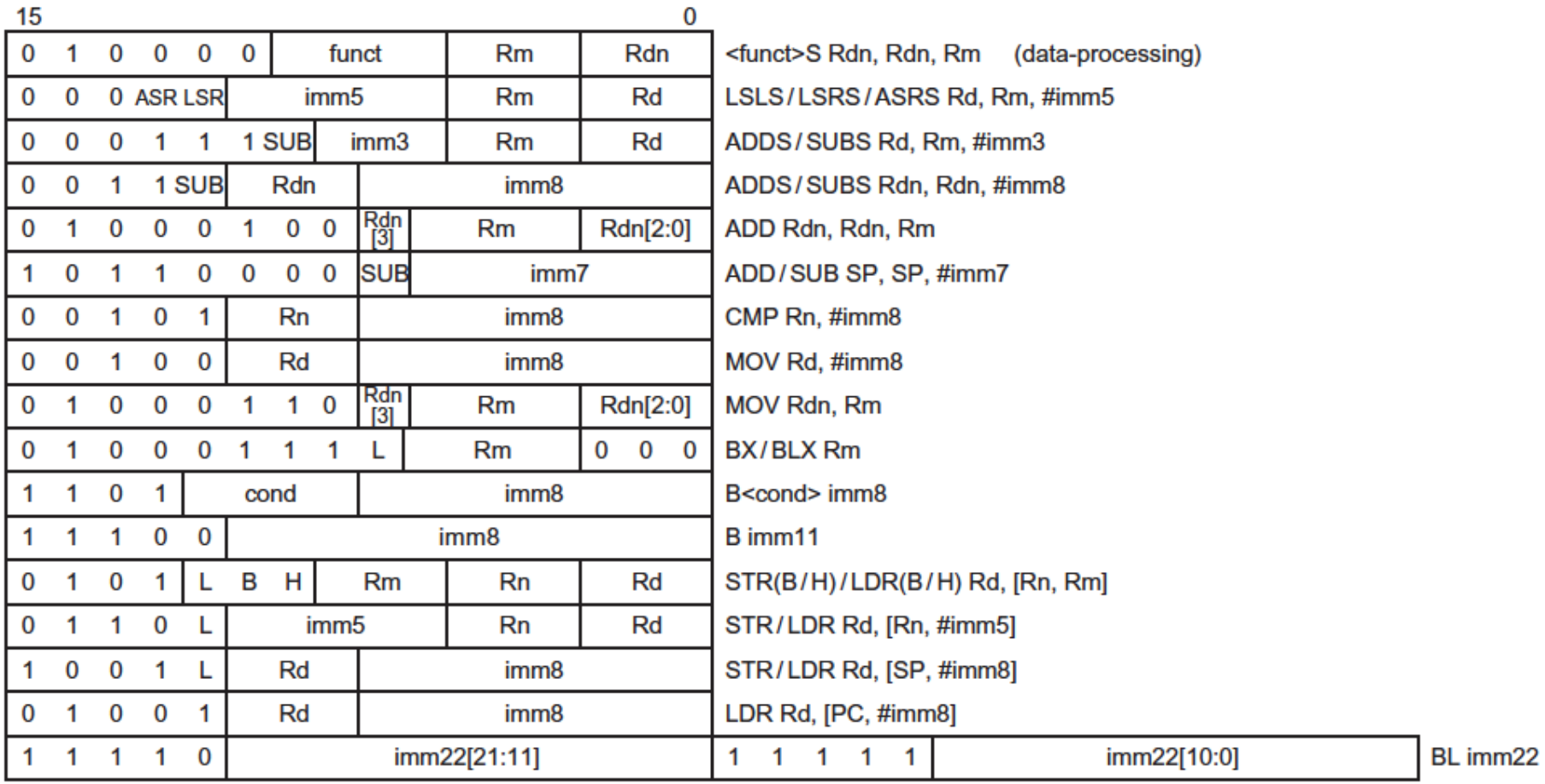
x86

ARM and Thumb Instructions

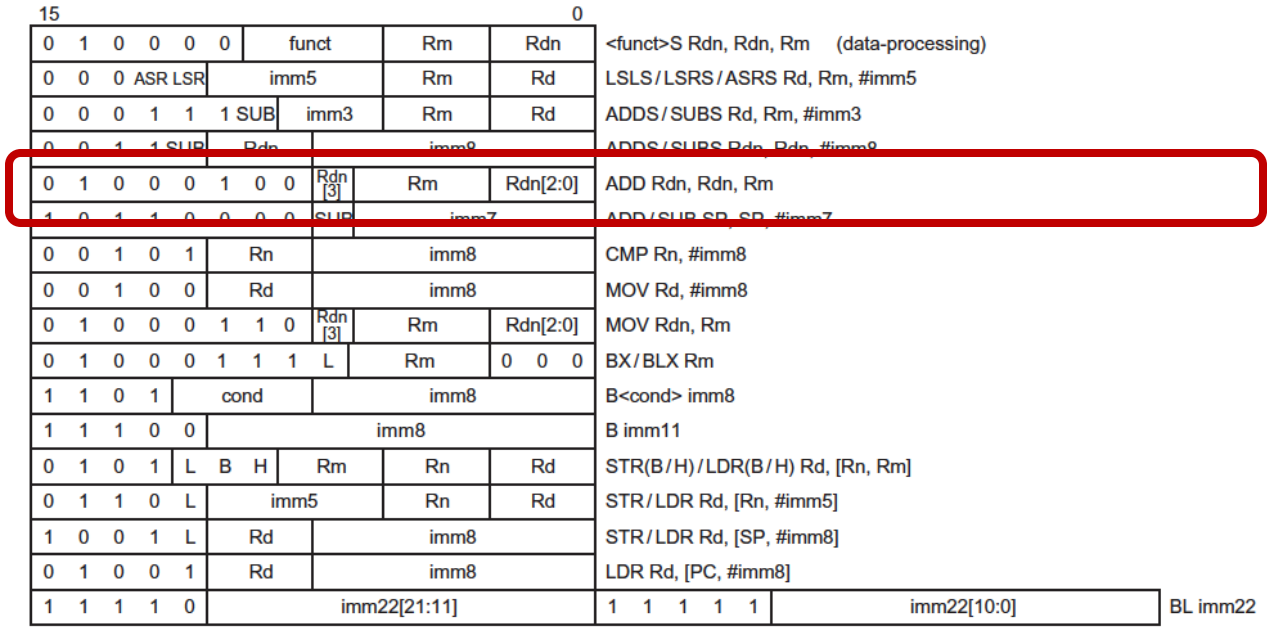
Lancaster University

- Thumb reduces program memory size and bandwidth requirements
  - Subset of instructions re-encoded into fewer bits (most 16 bits, some 32 bits)
    - Not all 32-bit instructions available
- 1995: Thumb-1: 16-bit instructions
- 2003: Thumb-2: Adds some 32-bit instructions, faster, low memory overhead.
- Some ARM CPU (not Cortex M3) allow control of the operating state
  - PSR T-bit: decodes instructions in Thumb state or ARM state
  - Thumb state indicated by program counter being odd (LSB = 1)
- **Microbit usesThumb instructions; always in Thumb state**
  - Some instruction will be 32-bit, because of space

ARM Thumb examples



Example Instruction – Add



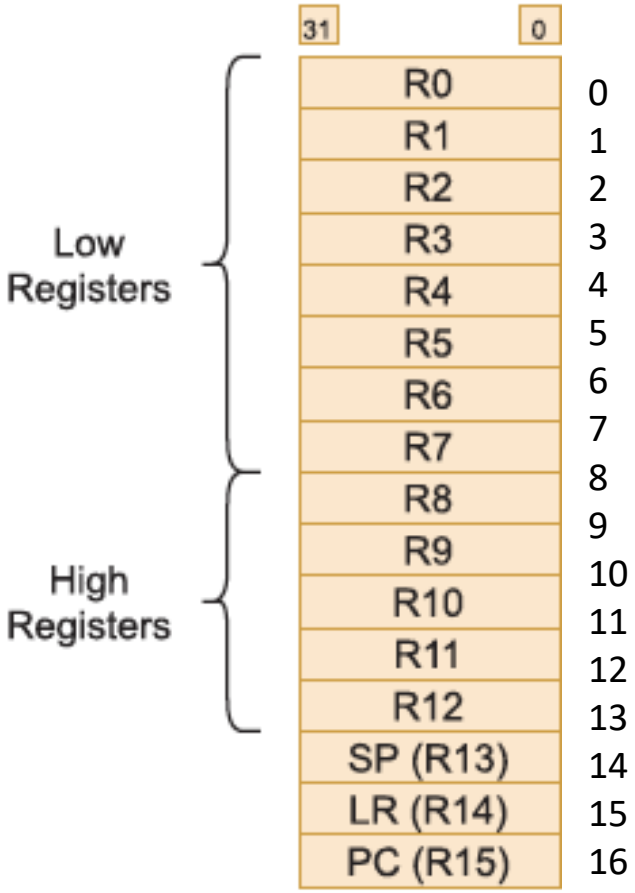
15															0	
0	1	0	0	0	0	funct						Rm		Rdn		<funct>S Rdn, Rdn, Rm (data-processing)
0	0	0	ASR	LSR		imm5						Rm		Rd		LSLS/LSRS/ASRS Rd, Rm, #imm5
0	0	0	1	1	1	SUB	imm3					Rm		Rd		ADDS/SUBS Rd, Rm, #imm3
0	0	1	1	SUB		Rdn				imm8						ADDS/SUBS Rdn, Rdn, #imm8
0	1	0	0	0	1	0	0	Rdn [3]		Rm		Rdn[2:0]				ADD Rdn, Rdn, Rm
1	0	1	1	0	0	0	0	SUB		imm7						ADD/SUB SP, SP, #imm7
0	0	1	0	1	Rn			imm8								CMP Rn, #imm8
0	0	1	0	0	Rd			imm8								MOV Rd, #imm8
0	1	0	0	0	1	1	0	Rdn [3]		Rm		Rdn[2:0]				MOV Rdn, Rm
0	1	0	0	0	1	1	1	L		Rm		0	0	0		BX/BLX Rm
1	1	0	1	cond				imm8								B<cond> imm8
1	1	1	0	0	imm8											B imm11
0	1	0	1	L	B	H	Rm			Rn		Rd				STR(B/H)/LDR(B/H) Rd, [Rn, Rm]
0	1	1	0	L	imm5					Rn		Rd				STR/LDR Rd, [Rn, #imm5]
1	0	0	1	L	Rd			imm8								STR/LDR Rd, [SP, #imm8]
0	1	0	0	1	Rd			imm8								LDR Rd, [PC, #imm8]
1	1	1	1	0	imm22[21:11]											BL imm22
1	1	1	1	1	imm22[10:0]											

```
add r2, r0
```

ARM Specifications

9 bits	1 bit	3 bits	3 bits
opcode	Rdn[3]	Rm	Rdn

Example Instruction – Add

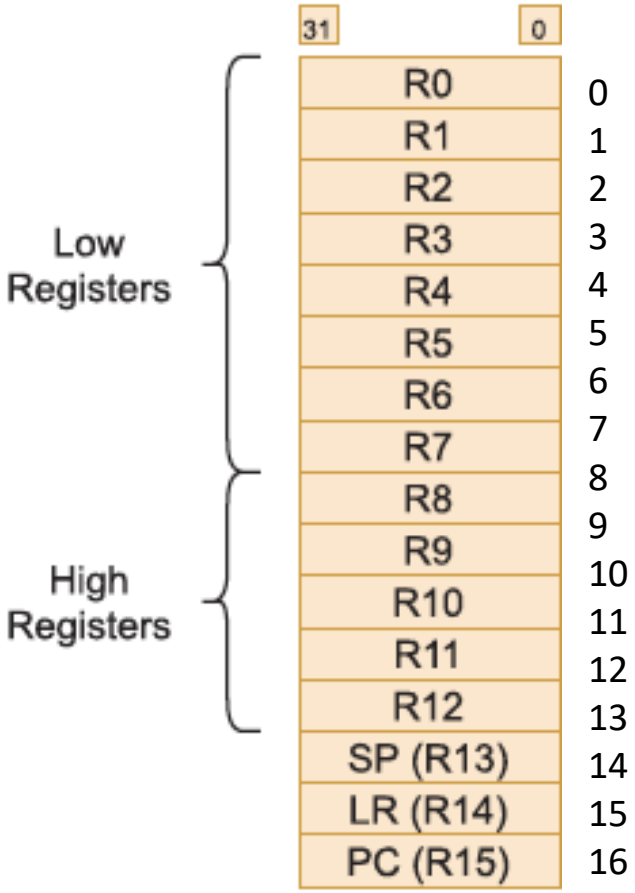



```
add r2, r0
```

9 bits	1 bit	3 bits	2 bits
01000100	Rdn[3]	Rm	Rdn
		0	2

9 bits	1 bit	3 bits	2 bits
01000100	0	000	10

Example Instruction – Add

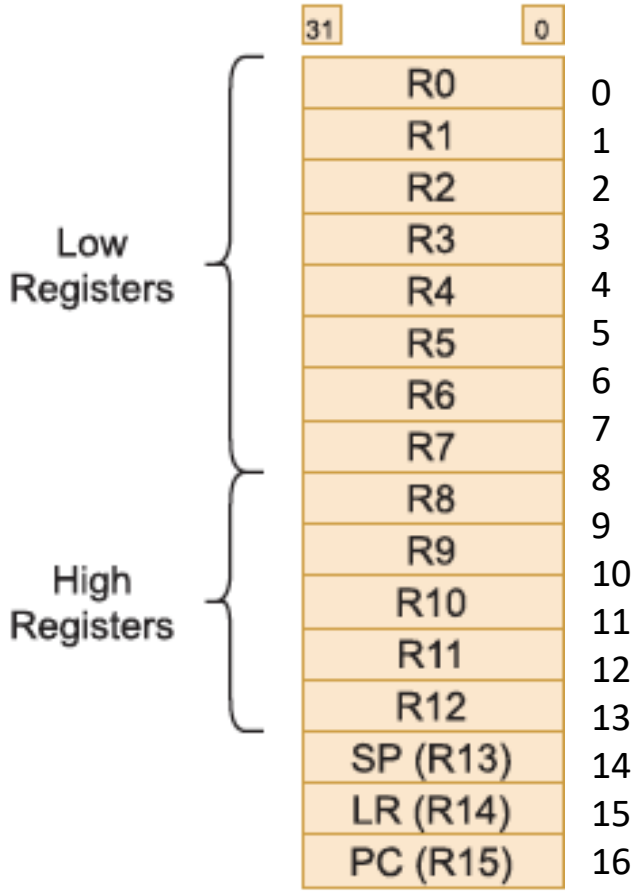



```
add r2, r0
```

9 bits	1 bit	3 bits	2 bits
01000100	Rdn[3]	Rm	Rdn
		0	2

9 bits	1 bit	3 bits	2 bits
01000100	0	000	10

Example Instruction – Add




```
add r2, r0
```

		0	2
9 bits	1 bit	3 bits	2 bits
01000100	0	000	10

0100 0100 0000 0010 (binary)

0x 4 4 0 2 (hex)

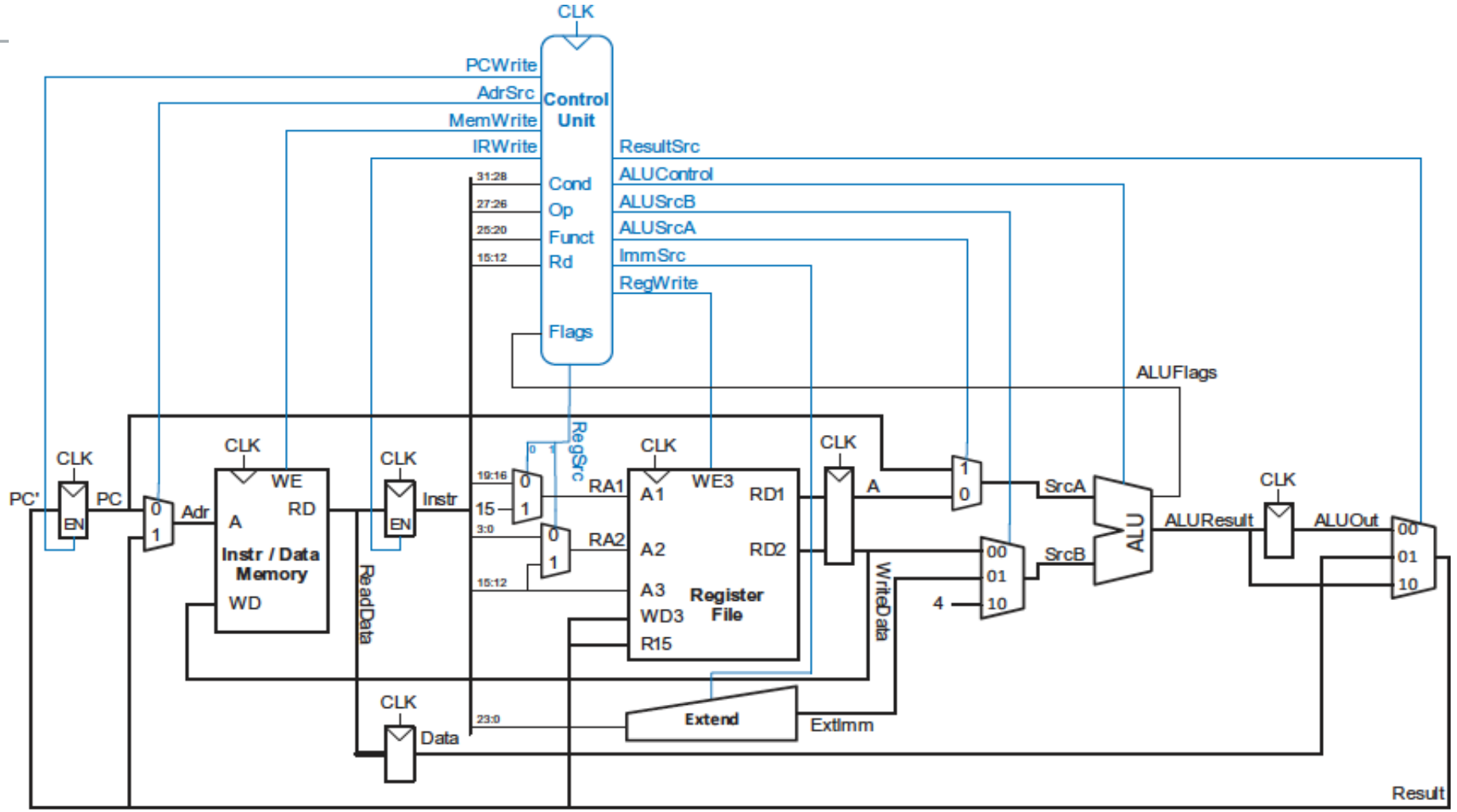
16-bit or 32-bit?

Lancaster University

- The CPU Control Unit reads instructions as a stream.
- The first 5-bits define if the instruction is 16 or 32 bit.
- If the value of bits[15:11] of the instruction is one of the following, the instruction is 32-bit long.
  - *0b11101* ⟶ `add r0, r1, r2`
  - *0b11110*
  - *0b11111*
- Otherwise, the halfword is a 16-bit instruction.

ARMv4 (Multi-Cycle) 32-bit

Lancaster University



Conclusion

Lancaster University

Machine code

- Moore's Law

ARM encoding

- ADD T32 and Thumb

NEXT: MMIO