M2 ARMv8 ISA Design

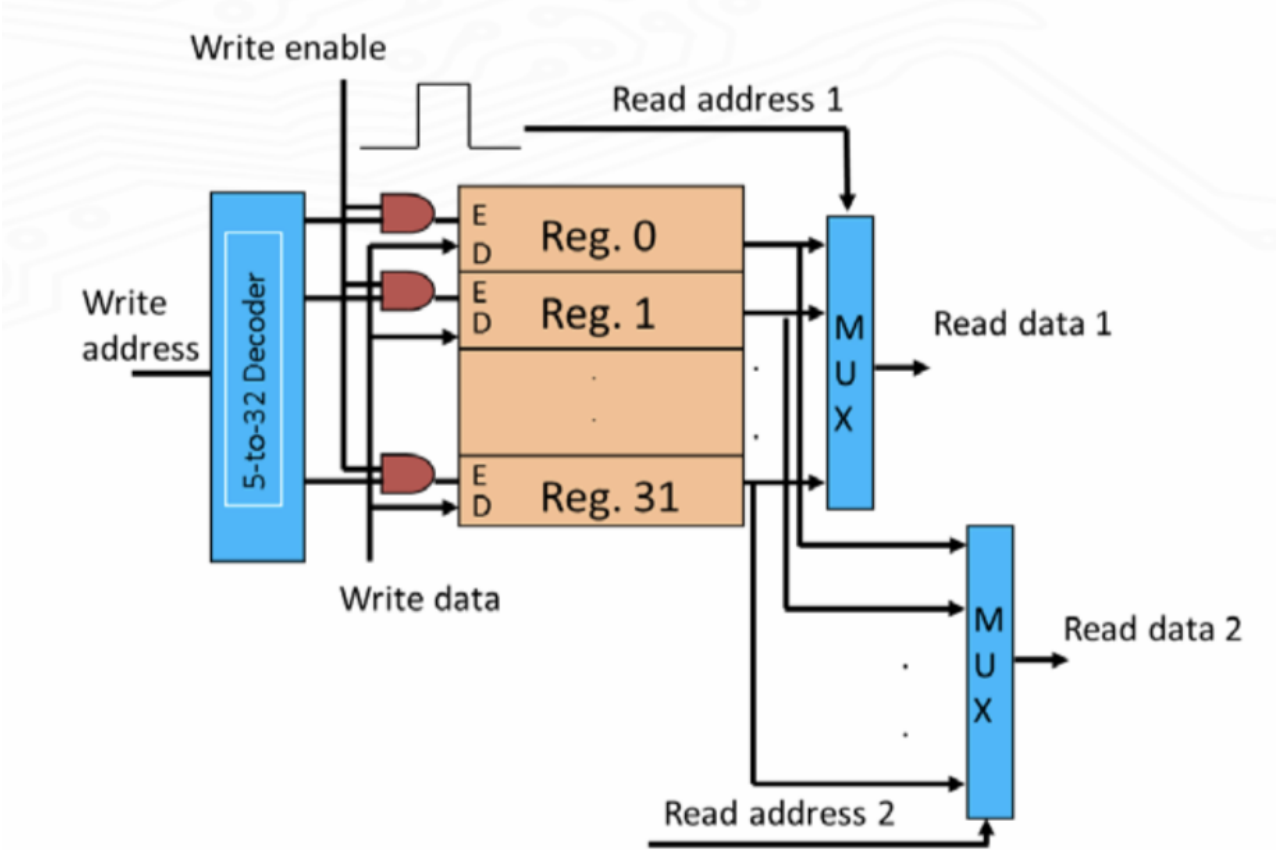
Things to note

- 64 bit address bus
- 64 bit data
- 32 bit instruction
- Size of register is 64 bits (8 bytes // doubleword)
- 32 register files

1. Design

1.1 Register specification

- Used for frequently accessed data
- **General Purpose Register** file
  - 32 x 64-bit registers
  - 2 read ports, 1 write port



- **Program Counter (PC)**
  - 64-bit register that holds (points to) the **address** of the next instruction

- **Register Usage Convention**

Name	Register number	Usage	Preserved on call?
X0 – X7	0-7	arguments/results	no
X8	8	Indirect result location register	no
X9-X15	9-15	Temporaries	no
X16(IP0)	16	May be used by linker as a scratch register; other times used as temporary register	no
X17(IP1)	17	May be used by linker as a scratch register; other times used as temporary register	no
X18	18	Platform register; other wise a temporary register	no
X19-X27	19-27	saved	yes
X28(SP)	28	Stack pointer	yes
X29(FP)	29	frame pointer	yes
X30(LR)	30	Link register (return address)	yes
X31	31	The constant value 0	yes

1.2 Memory Organisation (LEGv8 specific)

- Instruction size = 32 bits / 4 bytes
- Instruction is accessed one at a time so address of each instruction is a multiple of 4, starting from address 0x0..00
- Data size = 64 bits / 8 bytes
- Data is accessed one at a time so address of each data segment is a multiple of 8

2. Instructions

2.1 Classification

- Each instruction is 32 bits which contains information for Opcode, Destination / Source registers, Destination Address etc
    - Opcode: portion of machine language instruction that specifies the operation to be performed

Based on Functionality

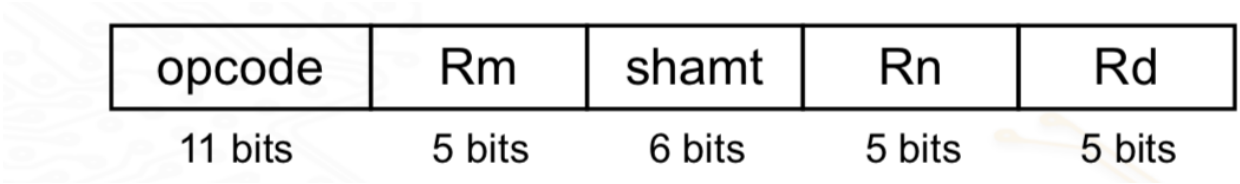
refer to slides if need clarification

- Arithmetic and Logical instructions
    - Arithmetic: ADD, SUB, ADDI, SUBI, and variants (ADDS, SUBS etc)
        - **Condition Codes** (flags) set from arithmetic instructions suffixed with S
            - N (negative): result has 1 in MSB (2s comp)

        - Z (zero): result is 0
        - C (carry): result has carryout from MSB
        - V (overflow): result overflowed
    - Logical: AND, ORR, EOR, ANDI, ORRI, EORI etc
    - Shift: LSL, LSR etc
- Data transfer instructions
    - LDUR, STUR, LDURB, STURB etc
- Conditional instructions
    - CBZ, CBNZ, B.cond etc
- Unconditional instructions
    - B, BR, BL etc

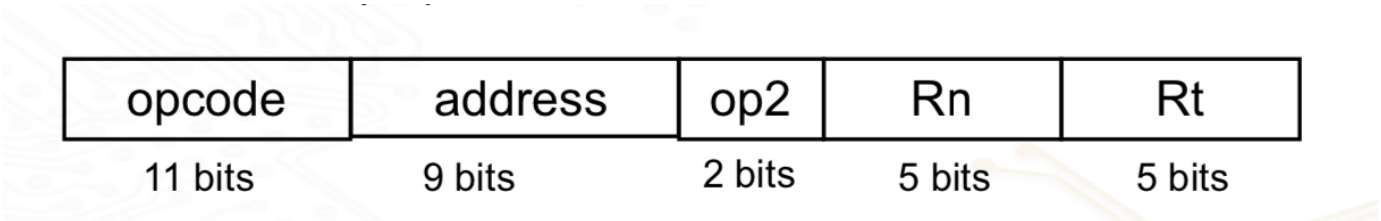
2.2 Instruction Format

Register (R) Type



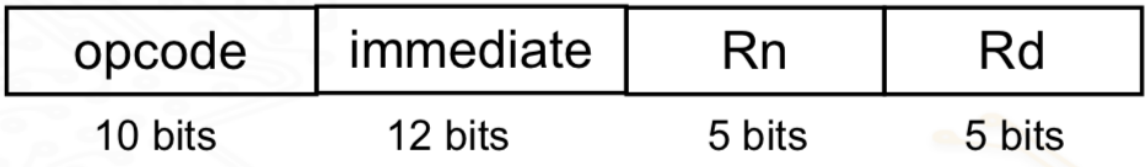
- `Rn` (first) and `Rm` (second) specifies source registers
- `Rd` specifies destination register
- `shamt` (shift amount), specifies number of bit positions to be shifted
- `opcode` specifies type of instruction

Data transfer (D) Type



- `Rn` specify base register (base register contains base address)
- `address` specifies offset from **contents** of base register
- `Rt` specifies destination register (for LD) or source register (for ST)

Immediate (I) Type (ADDI // Rd = Rd + Rn)

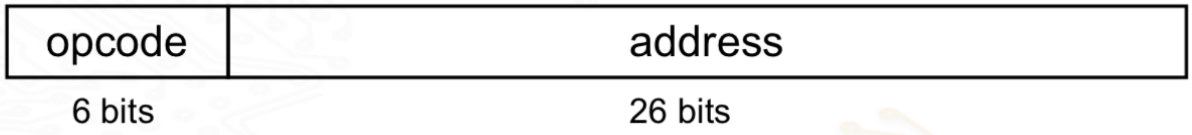


- `Rn` specifies source register
- `Rd` specifies destination register
- `immediate` is 0-extended
- `opcode` is 10 bits to allow more range for `immediate` field

Conditional Branch (CB) Type

- `address` specifies distance (number of instructions) to jump
  - Final address = PC + `address` * 4 (`address` shifts left 2 bytes)
- `Rt` specifies register to check condition

Unconditional Branch (B) Type



- `address` specifies the distance for PC to jump to (similar to CB Type)
  - Final address = PC + `address` * 4

3. Design Issues

3.1 Goals

- Simple microarchitecture implementation
- High performance
- Low power
- Programmability
- Compatible

3.2 Issues

- Selection of operations to be executed by instructions

- Operand locations
- Addressing mode of instruction formats
- Register size, word size, memory address space, address space

3.3 Design Policies

CISC (Complex Instruction Set Computer)

- Aims at a small program memory and less compiler workload
- Involves large instruction set comprised of complex and specialised instructions to have fewer instructions per task
- **Design features**
    - Source and Destination can be in registers / memory / both
    - Several addressing modes and multiple offset sizes
    - Microcoded instructions to realise pipelining, executed in multiple clock cycles
    - Relatively small program code size
    - Less compilation effort

RISC (Reduced Instruction Set Computer)

- 80% of instructions use only 20% of the instruction set
- All computing tasks can be performed by a small instruction set comprised of simple instructions (can be executed in a single clock cycle of short duration)
- **Design features**
    - Fixed Opcode-width and fewer addressing modes, fixed instruction length, fewer instruction formats with common fields
        - Advantages: simple and fast hardwired decoding and control generation, shorter clock period, load / store architecture
    - Only LD and ST and only one addressing mode for memory access; all operands and destination located in registers
        - Advantages: memory access and operand processing performed by separate instructions $\rightarrow$ faster register access
    - All instructions executed in a single cycle

CISC vs RISC

CISC	RISC
Many complex instructions in the instruction set	Fewer simple instructions in the instruction set
Many formats, & several addressing modes to support complex instructions	Fewer instruction formats, & a few addressing modes
Instruction length varies according to the addressing mode	Fixed instruction length) simpler implementation
Instructions are microcoded and executed in multiple clock cycles	Hardwired decoding: single cycle instruction execution
Memory can be referenced by many different modes	Only load/store instructions can reference memory
Operands could be memory: higher clock period : less number of registers	Operands in register for faster clocking: more registers : less memory access
Difficult to pipeline and super-scalar implementation	Easy to pipeline: and super-scalar implementation
Program code size is relatively small: complexity is in micro-program level	Program code size is usually large: complexity is in the compiler
Higher complexity of instruction implementation: CPI more than 1.	More compile time: higher register and more cache area