Embedded Systems

Instruction Set Architecture Design

Lesson 03

Francesco Menichelli

francesco.menichelli@uniroma1.it

Instruction Set Architecture (ISA)

The instruction set architecture is concerned with:

- Organization of programmable storage (memory & registers):

  Includes the amount of addressable memory and number of available registers.
- Data Types & Data Structures: Encodings & representations.
- Instruction Set: What operations are specified.
- Instruction formats and encoding.
- Modes of addressing and accessing data items and instructions
- Exception conditions.

Computer Instruction Sets

- Regardless of computer type, CPU structure, or hardware organization, every machine instruction must specify the following:
  - Opcode: Which operation to perform. Example: add, load, and branch.
  - Where to find the operand or operands, if any: Operands may be contained in CPU registers, main memory, or I/O ports.
  - Where to put the result, if there is a result: May be explicitly mentioned or implicit in the opcode.
  - Where to find the next instruction: Without any explicit branches, the instruction to execute is the next instruction in the sequence or a specified address in case of jump or branch instructions.

Operation Types in The Instruction Set

Operator Type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, or
Data transfer	Loads-stores (move on machines with memory addressing)
Control	Branch, jump, procedure call, and return, traps.
System	Operating system call/return, virtual memory management instructions ...
Floating Point	Floating point operations: add, multiply, divide
Decimal	Decimal add, decimal multiply, decimal to character conversion
String	String move, string compare, string search
Media	The same operation performed on multiple data (e.g. Intel MMX, SSE)

RISC vs. CISC

- **Reduced instruction set computer (RISC):**
  - load/store;
    - operands in memory must be first loaded into register before any operation
  - 32-bit fixed format instruction
  - 32 32-bit General Purpose Registers
    - (R0 contains zero, DP take pair of Registers)
  - 3-address, reg-reg arithmetic instruction
  - Single address mode for load/store: base + displacement
    - No indirection
  - Simple branch conditions
  - Examples: ARM, MIPS, Sun SPARC, PowerPC
- **Complex instruction set computer (CISC):**
  - many addressing modes;
    - can directly operate on operands in memory
  - many operations.
  - variable instruction length
  - Examples: Intel x86, 68000, VAX microprocessors and compatibles

Instruction Set Architecture: What Must be Specified?

- Instruction Format or Encoding
  - How is it decoded?
- Addressing of operands and result
  - Immediate addressing
  - Register addressing
  - Direct addressing
  - Register indirect addressing
  - Indexed addressing
  - Based-indexed addressing
  - PC Relative
  - Stack addressing
- Data type and Size
  - Fixed, Float, Vector
  - 8, 16, 32, 64 bit
  - Little Endian vs. Big Endian
- Operations
  - ADD, MUL, DIV, SHIFT, OR, AND
- Successor instruction
  - Jumps, conditions, branches
  - Fetch-decode-execute is implicit! –Von Neumann

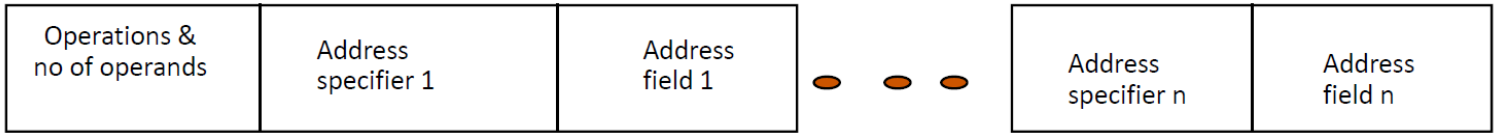
Instruction Format and Encoding

- **Considerations affecting instruction set encoding:**
  - The number of registers and addressing modes supported by ISA.
  - The impact of the size of the register and addressing mode fields on the average instruction size and on the average program.
  - To encode instructions into lengths that will be easy to handle in the implementation. On a minimum to be a multiple of bytes (8 bits).

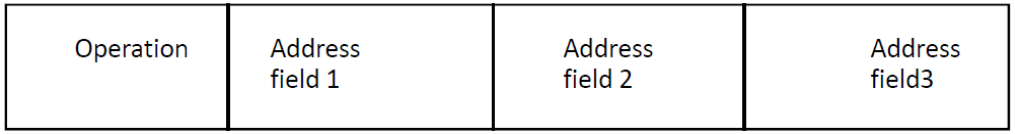
- **Instruction Encoding Classification:**
  - Fixed length encoding: Faster and easiest to implement in hardware.
  - Variable length encoding: Produces smaller instructions.
  - Hybrid encoding.

Three Examples of Instruction Set Encoding

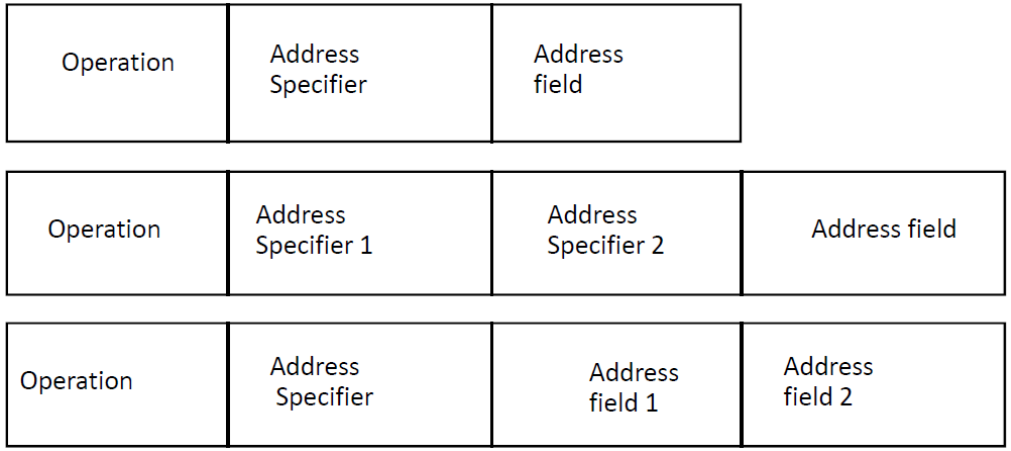
Variable Length Encoding: VAX (1-53 bytes)



Fixed Length Encoding: MIPS, PowerPC, SPARC (all instructions are 4 bytes each)



Hybrid Encoding: IBM 360/370, Intel 80x86



Types of ISAs According To Operand Addressing Fields

- **Memory-To-Memory Machines:**
  - Operands obtained from memory and results stored back in memory by any instruction that requires operands.
  - No local CPU registers are used in the CPU datapath.
  - Include:
    - The 4-Address ISA.
    - The 3-Address ISA.
    - The 2-Address ISA.
- **The 1-address (Accumulator) ISA:**
  - A single local CPU special-purpose register (accumulator) is used as the source of one operand and as the result destination

Types of ISAs According To Operand Addressing Fields

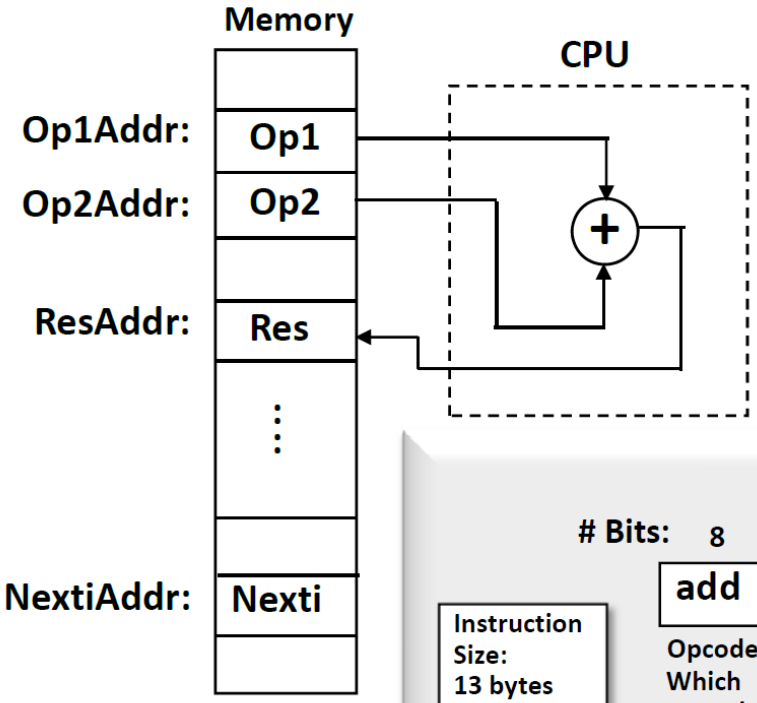
- **The 0-address or Stack ISA:**
  - A push-down stack is used in the CPU.

- **General Purpose Register (GPR) ISA:**
  - The CPU datapath contains several local general-purpose registers which can be used as operand sources and as result destinations.
  - A large number of possible addressing modes.
  - Load-Store or Register-To-Register Machines: GPR machines where only data movement instructions (loads, stores) can obtain operands from memory and store results to memory.

Types of ISAs -Memory-To-Memory Machines: The 4-Address ISA

No program counter (PC) or other CPU registers are used.
Instruction encoding has four address fields to specify:

- Location of first operand, location of second operand.
- Place to store the result, location of next instruction.



Instruction:

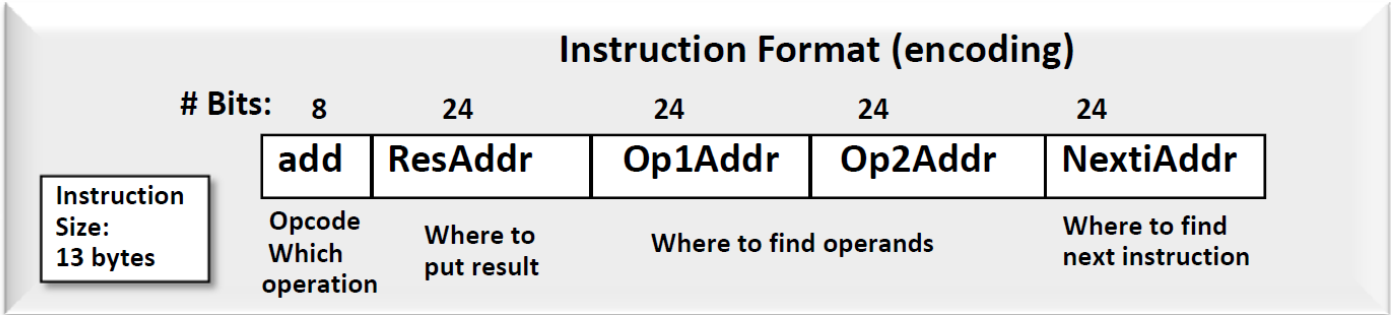
add Res, Op1, Op2, Nexti

Meaning:

Res <- Op1 + Op2

or more precise RTN:

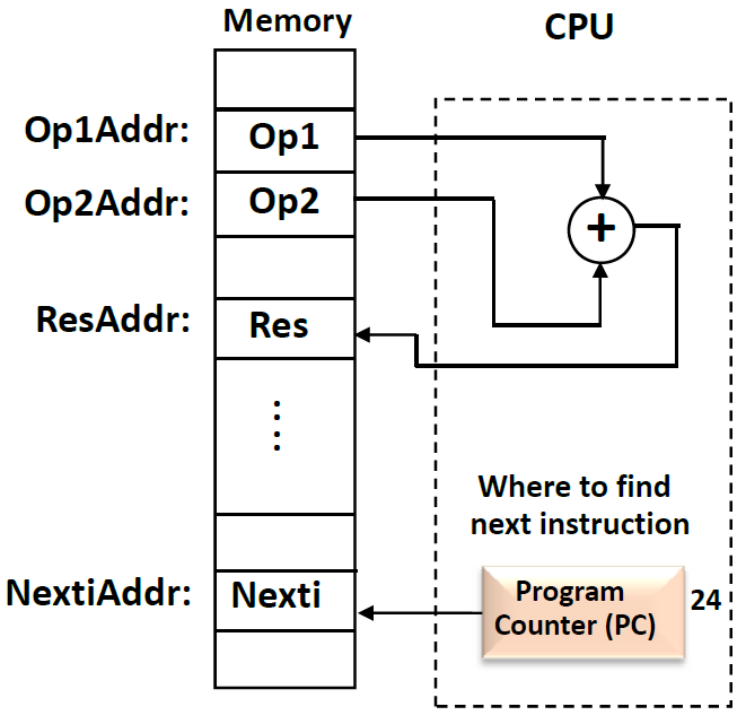
M[ResAddr] <- M[Op1Addr] + M[Op2Addr]



Types of ISAs -Memory-To-Memory Machines: The 3-Address ISA

A program counter (PC) is included within the CPU which points to the next instruction.

No CPU storage (general-purpose registers).



Instruction:

sub Res, Op1, Op2

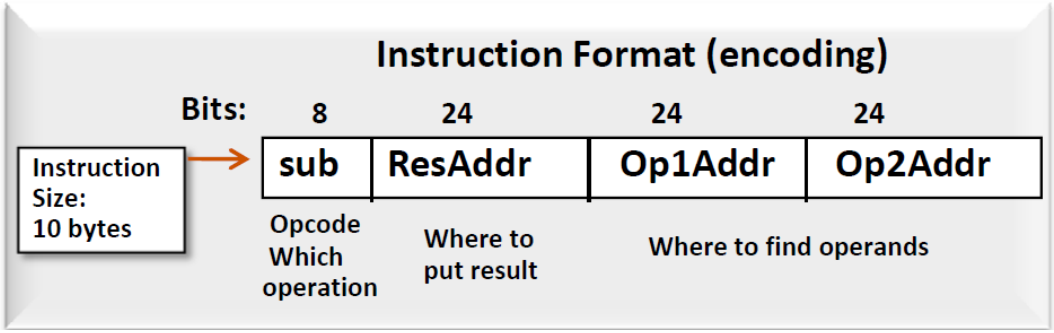
Meaning:

Res <- Op1 - Op2

or more precise RTN:

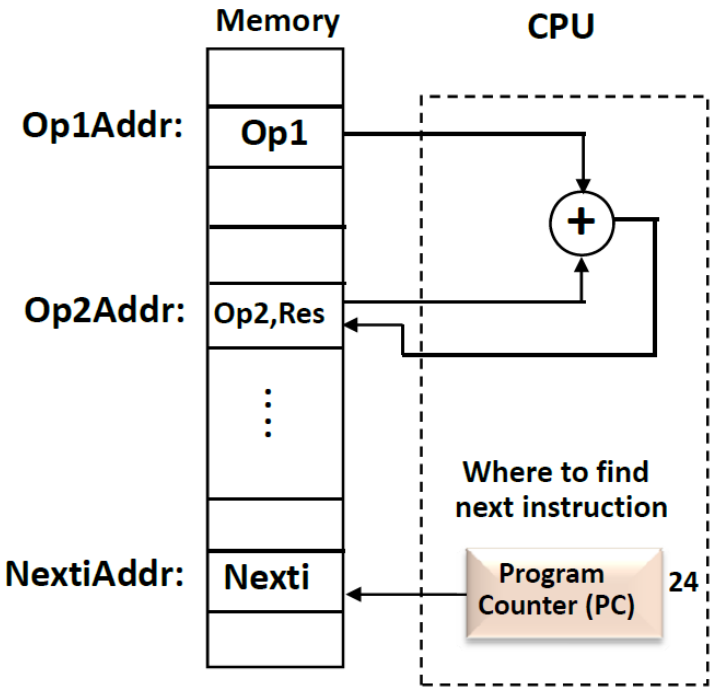
M[ResAddr] <- M[Op1Addr] - M[Op2Addr]

PC <- PC + 10 ← Increment PC



Types of ISAs -Memory-To-Memory Machines: The 2-Address ISA

The 2-address Machine: Result is stored in the memory address of one of the operands.



Can address 2^{24} bytes = 16 MBytes

Instruction:

add Op2, Op1

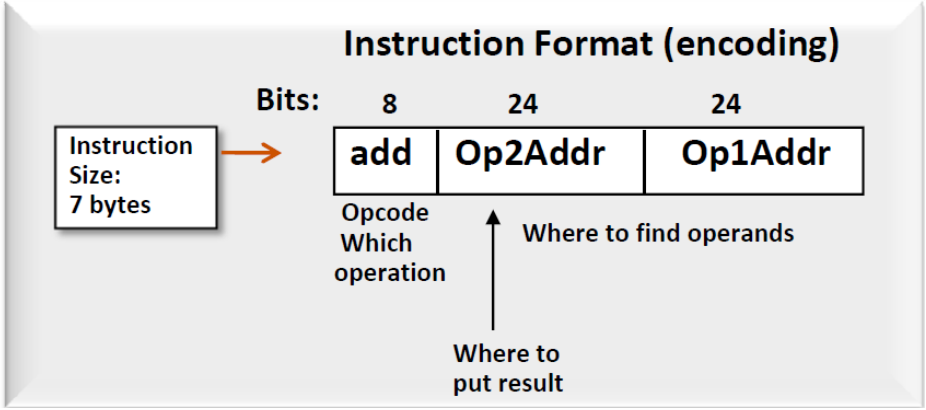
Meaning:

Op2 <- Op1 + Op2

or more precise RTN:

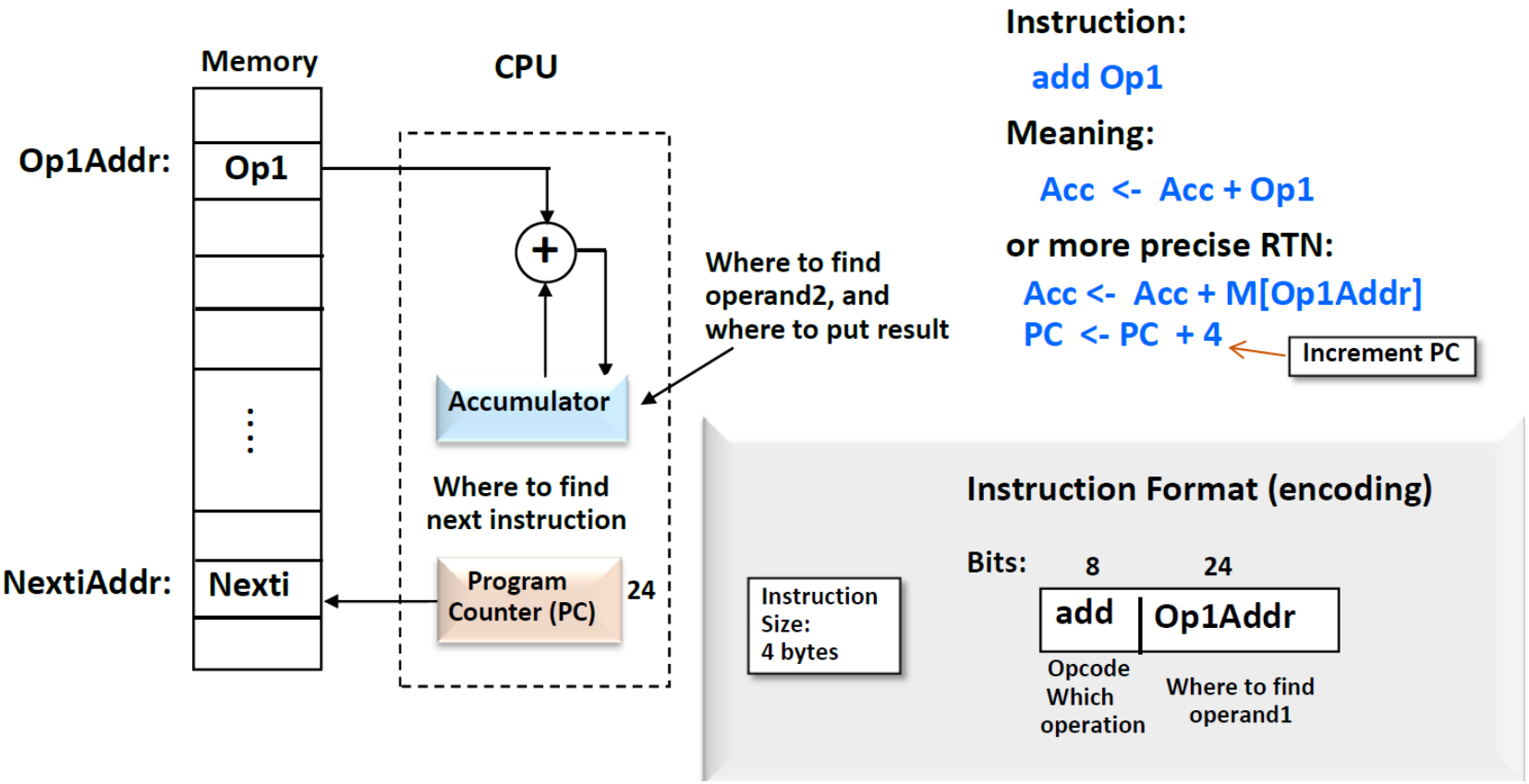
M[Op2Addr] <- M[Op1Addr] + M[Op2Addr]

PC <- PC + 7 ← Increment PC



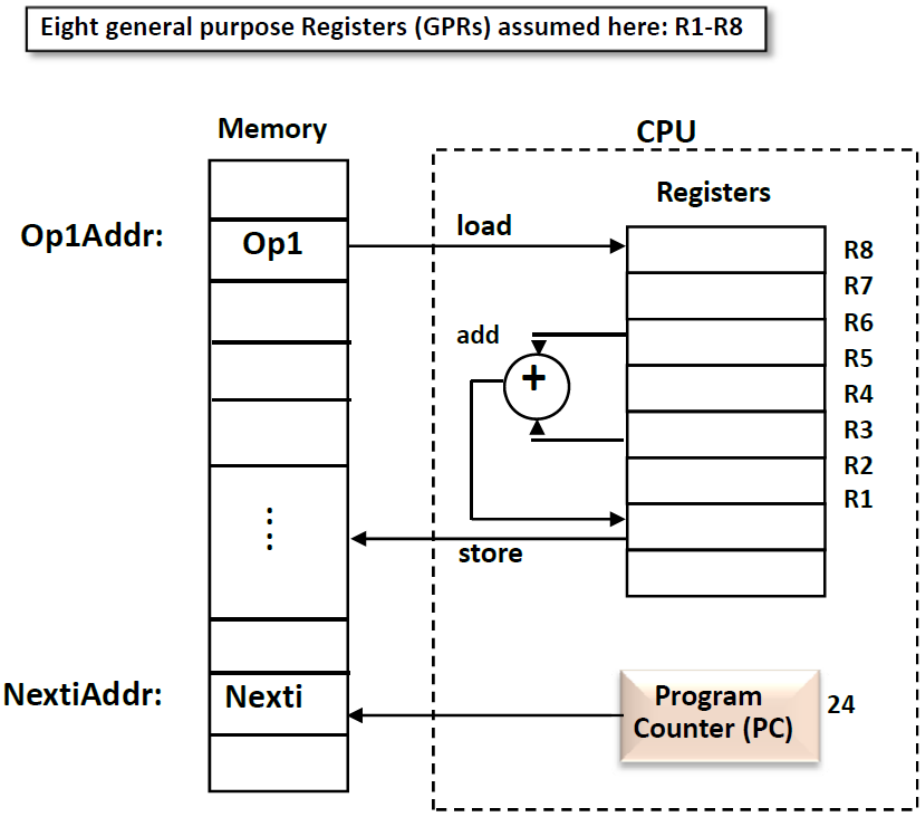
Types of ISAs -Memory-To-Memory Machines: The 1-Address(Accumulator) ISA

A single accumulator in the CPU is used as the source of one operand and result destination.



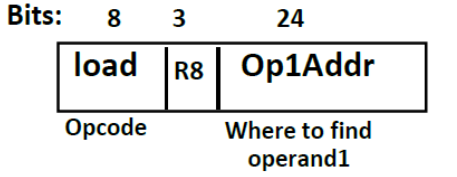
Types of ISAs General Purpose Register (GPR) Machines

CPU contains several general-purpose registers which can be used as operand sources and result destination.



Instruction:
load R8, Op1
Meaning:
R8 <- M[Op1Addr]
PC <- PC + 5

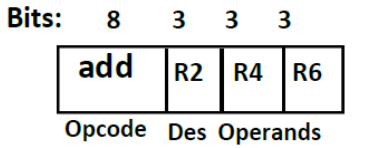
Instruction Format



Size = 4.375 bytes rounded up to 5 bytes

Instruction:
add R2, R4, R6
Meaning:
R2 <- R4 + R6
PC <- PC + 3

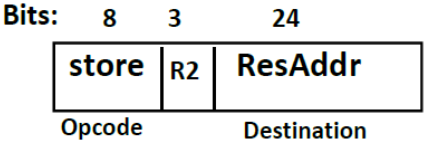
Instruction Format



Size = 2.125 bytes rounded up to 3 bytes

Instruction:
store R2, Op2
Meaning:
M[Op2Addr] <- R2
PC <- PC + 5

Instruction Format



Size = 4.375 bytes rounded up to 5 bytes

The add instruction has three register specifier fields
While load, store instructions have one register specifier field and one memory address specifier field

Expression Evaluation Example with 3-, 2-, 1-Address, and GPR Machines

For the expression A = (B + C) * D-E where A-E are in memory

3-Address	2-Address	1-Address Accumulator	GPR Register-Memory	GPR Load-Store

add A, B, C
mul A, A, D
sub A, A, E
	load A, B
	add A, C
	mul A, D
	sub A, E
		load B
		add C
		mul D
		sub E
		store A
			load R1, B
			add R1, C
			mul R1, D
			sub R1, E
			store A, R1
				load R1, B
				load R2, C
				add R3, R1, R2
				load R1, D
				mul R3, R3, R1
				load R1, E
				sub R3, R3, R1
				store A, R3

3 instructions Code size: 30 bytes 9 memory accesses for data	4 instructions Code size: 28 bytes 11 memory accesses for data	5 instructions Code size: 20 bytes 5 memory accesses for data	5 instructions Code size: 25 bytes 5 memory accesses for data	8 instructions Code size: 34 bytes 5 memory accesses for data

Instruction Set Architecture Tradeoffs

- 3-address machine: shortest code sequence; a large number of bits per instruction; large number of memory accesses.
- General purpose register machine (GPR):
  - Addressing modified by specifying among a small set of registers with using a short register address (all new ISAs since 1975).
  - Advantages of GPR:
    - Low number of memory accesses. Faster, since register access is currently still much faster than memory access.
    - Registers are easier for compilers to use.
    - Shorter, simpler instructions.
- Load-Store Machines: GPR machines where memory addresses are only included in data movement instructions (loads/stores) between memory and registers.

Instruction Set Architecture Tradeoffs

- Accumulator Machines
  - Requires storing lots of temporary and intermediate values in memory
  - Accumulator is only really beneficial for a chain (sequence) of calculations where the result of one is the input to the next.
- Memory-to-Memory ISAs
  - Main memory much slower than arithmetic circuits
    - This was as true in 1950 as in 2015!
  - It takes a lot of room to specify memory addresses
  - Results are often used one or two instructions later

Typical GPR ISA Memory Addressing Modes

Addressing mode	Sample Instruction	Meaning
Register	add R4, R3	R4 <-R4 + R3
Immediate	add R4, #3	R4 <-R4 + 3
Displacement	add R4, 10 (R1)	R4 <-R4 + Mem[10+ R1]
Indirect	add R4, (R1)	R4 <-R4 + Mem[R1]
Indexed	add R3, (R1 + R2)	R3 <-R3 +Mem[R1 + R2]
Absolute	add R1, (1001)	R1 <-R1 + Mem[1001]
Memory indirect	add R1, @ (R3)	R1 <-R1 + Mem[Mem[R3]]
Autoincrement	add R1, (R2) +	R1 <-R1 + Mem[R2] R2 <-R2 + d
Autodecrement	add R1, -(R2)	R2 <-R2 -d R1 <-R1 + Mem[R2]
Scaled	add R1, 100 (R2) [R3]	R1 <-R1+ Mem[100+ R2 + R3*d]

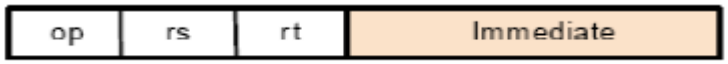
"d" is generally the size of the memory word width e.g. 4 bytes

Addressing Modes

- **Register addressing:**
  - Specify register number rather than address.

    MOV R1, R2
  - Copy content of register 2 to register 1.
- **Immediate addressing:**
  - Address part of operand contains operand itself.

    MOV R1, #4 -> Load constant 4 to register 1.
  - Only small integer constants can be specified in this way.
  - No memory reference to fetch data
  - Fast
  - Limited range



Addressing Modes

- **Register indirect addressing:**
  - Operand address is not contained in instruction but in a register.
  - Operand address is a pointer.

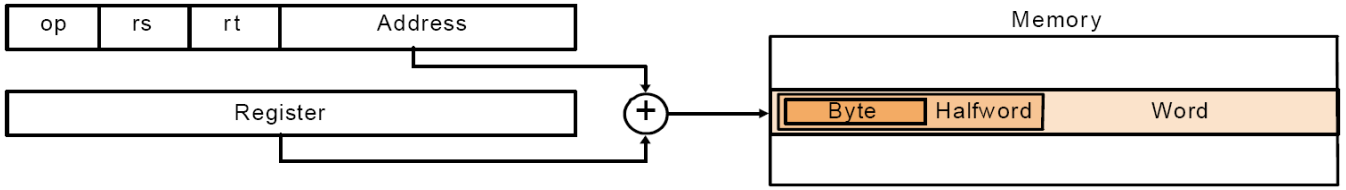
    ADD R1, (R2); add to register R1 word at address contained in R2.
  - Can refer to different addresses in different instruction.
    - Example: assembly code for adding the elements of an array.

```
        MOV R1, #0       ; accumulate sum in R1, initially 0
        MOV R2, #A       ; R2 = address of the array A
        MOV R3, #A+4096  ; R3 = address of first word beyond A
LOOP:   ADD R1, (R2)     ; register indirect through R2 to get operand
        ADD R2, #4       ; increment R2 by one word (4 bytes)
        CMP R2, R3       ; are we done yet?
        BLT LOOP         ; if R2 < R3, we are not done, so continue
```

Addressing Modes

- **Indexed addressing:**
  - Memory is addressed by giving a register plus a constant offset.
  - Example: processing of static arrays.

MOV R4, A(R2); load into R1 word whose address has offset A from content of R2.



Array is at a fixed address; register contains current index.

- Example: assembly code for computing: $\sum_i A_i * B_i$.

```
        MOV R1, #0     ; accumulate the sum in R1, initially 0
        MOV R2, #0     ; R2 = index i
        MOV R3, #4096  ; R3 = first index value not in use
LOOP:   MOV R4, A(R2)  ; R4 = A[i]
        MUL R4, B(R2)  ; R4 = A[i] * B[i]
        ADD R1, R4     ; sum all the products into R1
        ADD R2, #4     ; i = i+4 (1 word = 4 bytes)
        CMP R2, R3     ; are we done yet?
        BLT LOOP       ; if R2 < R3, we are not done, so continue
```

Addressing Modes

- **Based-Indexed Addressing**
  - Address is computed by sum of two registers plus optional offset.
  - Processing of dynamic arrays.

    MOV R4, (R2+R5); load into R4 word whose address is the sum of R2 and R5.
  - R5 is the base address of the array.
  - R2 is the current index.
  - Replace loop code in previous example as follows:

```
        ...
        MOV R5, #A        ; R5 = address of A
        MOV R6, #B        ; R6 = address of B
LOOP:   MOV R4, (R2+R5)   ; R4 = A[i]
        MUL R4, (R2+R6)   ; R4 = A[i] * B[i]
        ...
```

Addressing Modes

- **Memory Indirect**
  - Memory cell pointed to by address field contains the address of (pointer to) the operand

    EA = (A) ; Look in A, find address (A) and look there for operand

    e.g. ADD (A)

    Add contents of cell pointed to by contents of A to accumulator
  - May be nested, multilevel, cascaded

    e.g. EA = (((A)))
  - Multiple memory accesses to find operand
  - Very slow

Addressing Modes for Branch Instructions

- **How to specify target address of branch instructions/procedure calls?**
  - Direct addressing: unconditional branches (gotos).
    - Generated from conditionals and loops.
  - Register indirect addressing or indexed mode.
    - Program may compute target address (computed goto, switch).
  - PC-relative addressing: indexed mode where PC acts as register.
    - Target address is specified as offset to current instruction.

- **Modes presented so far are also useful for branch instructions**

Considerations in designing an ISA

- **Architectural State: Memory and Registers**
  - Abstraction used by both compiler and Microarchitecture
  - Register specialization?
  - RISC and VLIW tend to use a large pool of general-purpose registers
  - Implementation techniques provide illusion of large number of registers or fast flat memory space (register renaming)

- **Pipelining and operational latency**
  - Sequential execution vs. pipelined execution model
  - Exposed or hidden pipelining (latency of operations): delayed branches, bypass networks, scoreboard
  - DSPs enforce uniform latency model

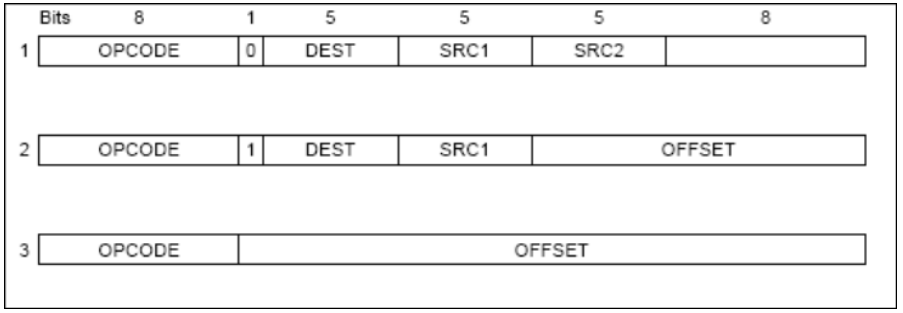
- **Delayed branches expose part of the RISC pipeline**

Considerations in designing an ISA

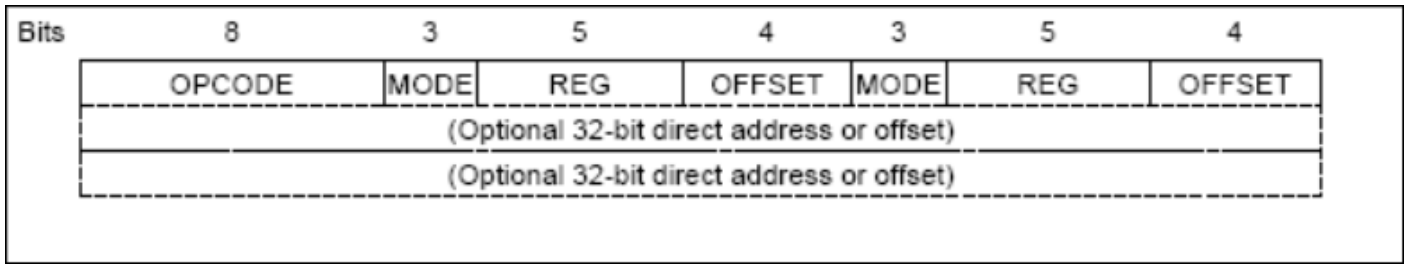
- **Encoding and Architectural Style:**
  - Architecture description contains architectural state, instruction set, and execution model
    - Implementations are free to build whatever structure they want, as long as they simulate the architectural model
  - RISC Encodings
    - Single instruction corresponds to a single operation
    - 1:1 encoding (Instruction encoding size: Number of run-time operations)
  - CISC Encodings
    - RISC subset plus more complex instructions
    - Variable: Dynamic encoding
  - VLIW Encodings
    - n:n; Fallacy: VLIW Instructions Have Fixed Width
  - DSP Encodings
    - 1:n
  - Vector Encodings1:variable

Considerations in designing an ISA

- **In a clean design, every opcode should permit every addressing mode.**
  - Three Address Machine
    - Two formats selected by bit.
    - 1 special format for branches



  - Two Address Machine
    - Each operand specified by 12 bits.
    - Mode, register, offset.
    - Optional 32-bit word for address.



Summary

- **Programmable CPU cores are important for the control parts of typical embedded application.**
- **They are well supported with tools to support the development of end-user software. (vs. deeply embedded SW)**
- **“Keep it Simple” heuristic (RISC vs. CISC)**
  - Make frequent cases fast and rare cases correct.
  - Regular (orthogonal) instruction set
  - No special features that match a high level language construct.
  - At least 16 registers to ease register allocation.
- **Embedded cores are often light cores which are a compromise between performance, area and power dissipation. (vs. stand-alone CPU cores which are optimized for performance)**

CPU in embedded systems

- Main constraints in embedded systems
  - Cost (Area)
  - Performance
  - Power consumption
- They are not independent
  - Performance/Cost
  - Performance/Power consumption

CPU in embedded systems

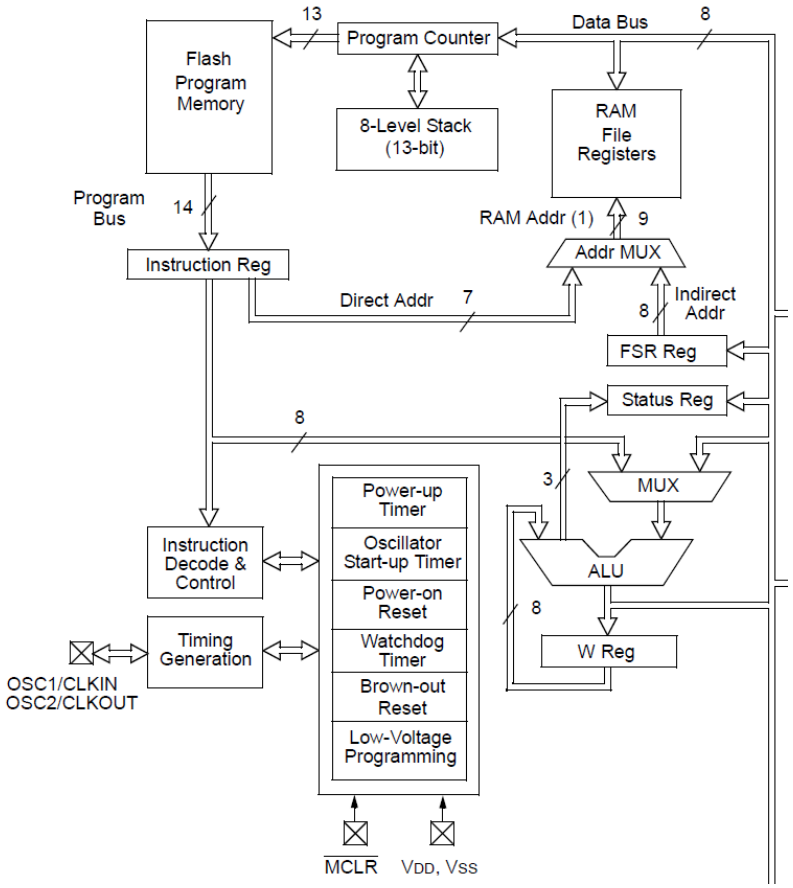
- Very wide range of applications
  - disposable electronic, consumer, automotive, Safety, Space
- Can not be achieved by a single or a reduced set of architectures
  - 4 bit architectures (very low end)
  - 8bit – 32bit (the most common)
  - 16bit, DSP, VLIW (specialized applications)
- The same constraints impact the whole architecture
  - Bus width
  - Memory subsystem (flash, ram, cache)
  - Power management

CPU in embedded systems

- The result is a multitude of components and architectures
  - Microchip: PIC16, PIC18, PIC24, dsPIC, PIC32
  - ST Microelectronics: ST6, ST7, ST9, ST10, STM8, STM32
- The same can be repeated for Atmel, TI (Texas Instruments), Infineon, NXP, etc.

Microchip PIC16F architecure

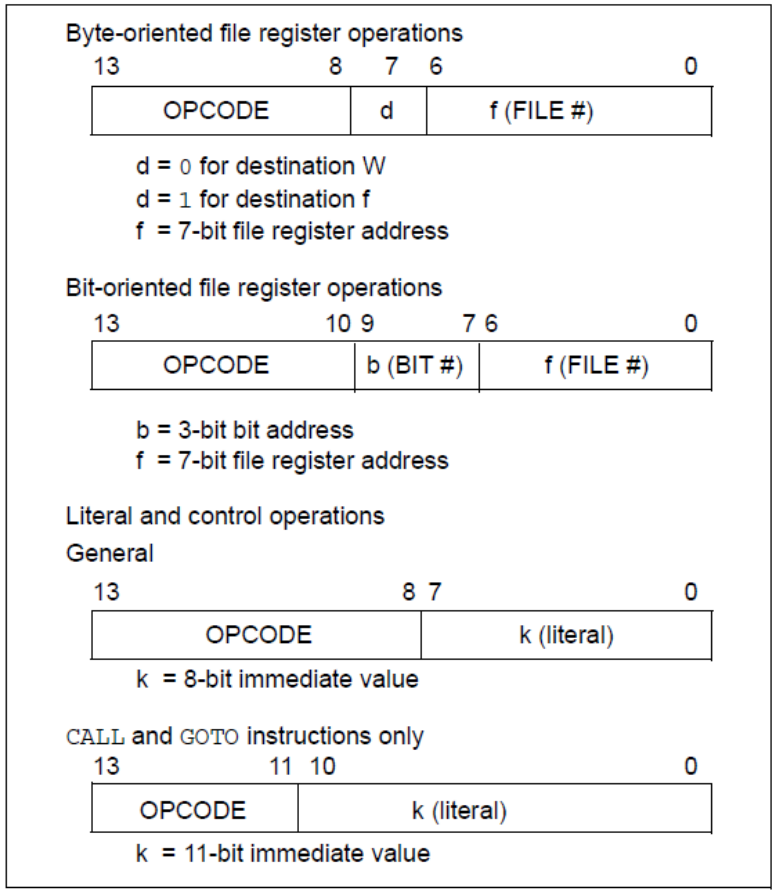
- Very low cost, in-house design (no compatibility with other CPUs)



- Strict Harvard architecture
- “File registers” RAM (64-256byte)
- Two stage pipeline (fetch, execute)
- 14-bit instruction set (14-bit instruction memory)
- 9-bit memory addressing (banked)
- 8-bit integer ALU
- Single data register (W)
- Very limited Stack (8-Level, PC only)

Microchip PIC16F instruction format

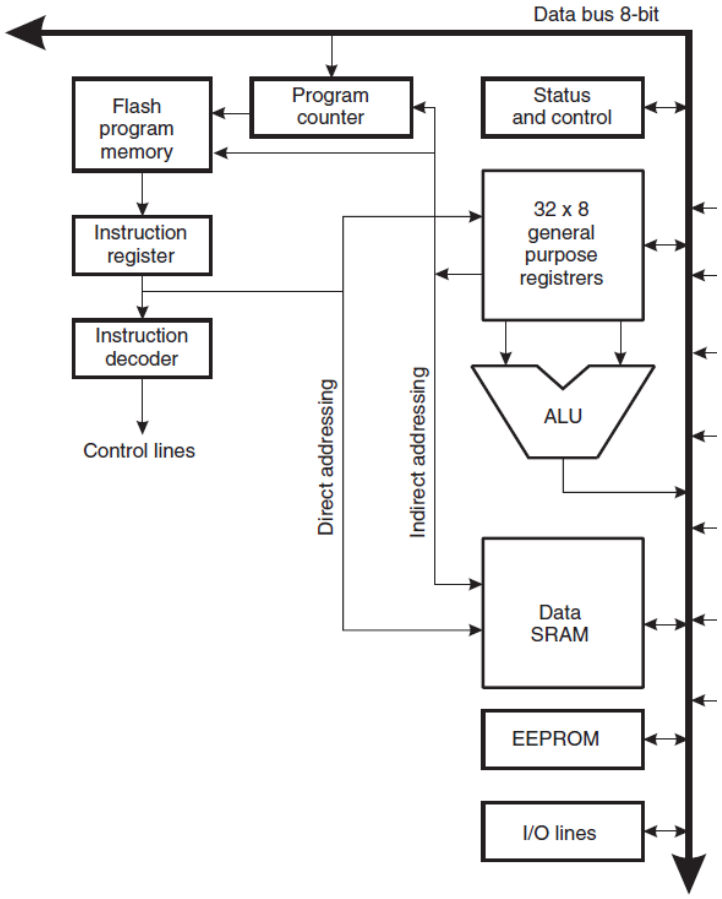
FIGURE 15-1: GENERAL FORMAT FOR INSTRUCTIONS



- 14 bit instruction set
- Accumulator based architecture (1-address ISA)
- Destination selectable (accumulator, memory)
- 4 instructions formats
  - byte oriented (MOV, ADD, AND, etc.)
  - bit oriented (set/reset bits)
  - literal (immediate accumulator loading)
  - Control flow (call, jump)

ATMEL AVR8 architecture

- Low cost, in-house design (no compatibility with other CPUs)



- Harvard architecture
- data SRAM (1-4Kbyte)
- Two stage pipeline (fetch, execute)
- 16-bit instruction set (16-bit instruction memory)
- 16-bit memory addressing (linear)
- 8-bit integer ALU (two independent operands)
- 32 data registers (8-bit)
- RAM Stack (stack pointer)

ATMEL AVR8 architecture

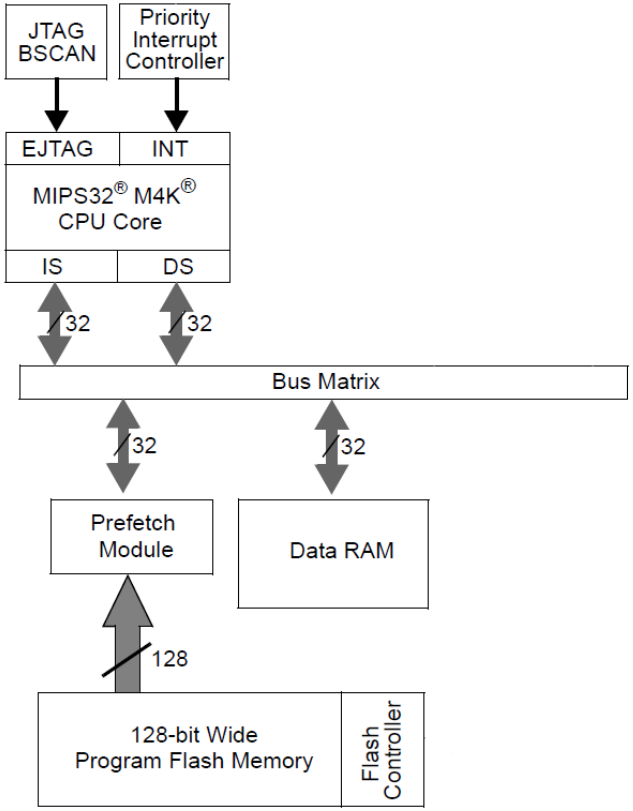
- AVR CPU general purpose working registers

	7 0	Addr.	
	R0	0x00	
	R1	0x01	
	R2	0x02	
	…		
	R13	0x0D	
General	R14	0x0E	
Purpose	R15	0x0F	
Working	R16	0x10	
Registers	R17	0x11	
	…		
	R26	0x1A	X-register Low Byte
	R27	0x1B	X-register High Byte
	R28	0x1C	Y-register Low Byte
	R29	0x1D	Y-register High Byte
	R30	0x1E	Z-register Low Byte
	R31	0x1F	Z-register High Byte

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Microchip PIC32 architecture

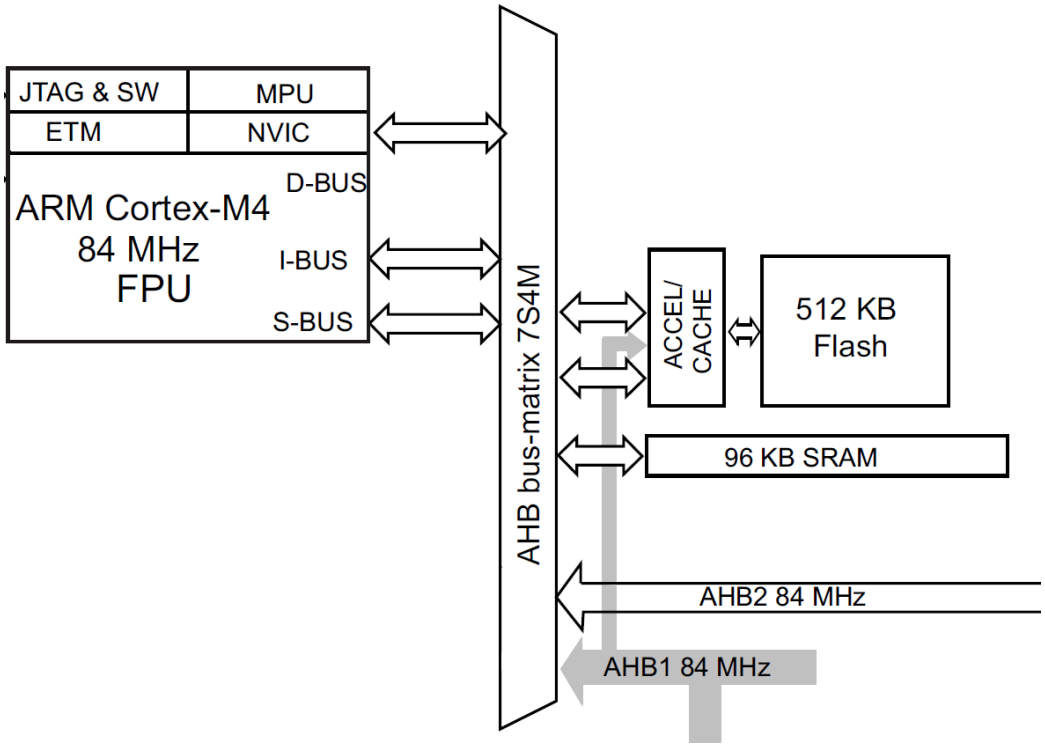
- MIPS32 M4K (MIPS Technologies)



- Harvard architecture (cache level)
- data SRAM (128-256Kbyte)
- five stage pipeline
- 32-bit instruction set (32-bit wide data memory)
- 128-bit wide program memory+prefetch
- 32-bit memory addressing (linear)
- 32-bit integer ALU (two independent operands+result)
- 32 data registers (32-bit)
- RAM Stack (stack pointer)

STM32 architecture

- ARM Cortex-M4 (ARM Holdings plc): ARMv7E-M architecture



- Harvard architecture (without cache)
- Separated D-BUS, I-BUS, S-BUS
- data SRAM
- five stage pipeline
- 16/32-bit instruction set (THUMB-2 instruction set)
- 32-bit memory addressing (linear)
- 32-bit integer ALU (two independent operands+result)
- 16 data registers (32-bit)
- RAM Stack (stack pointer)