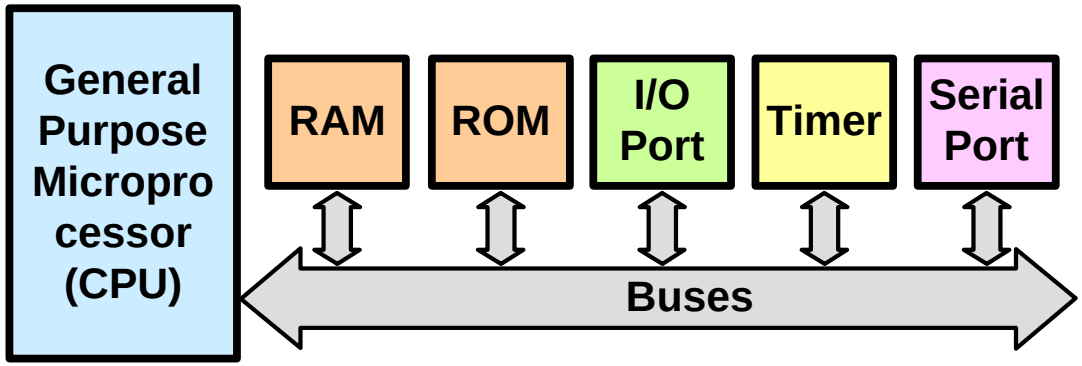
Introduction to STM32 and Arm Microcontrollers

Topics

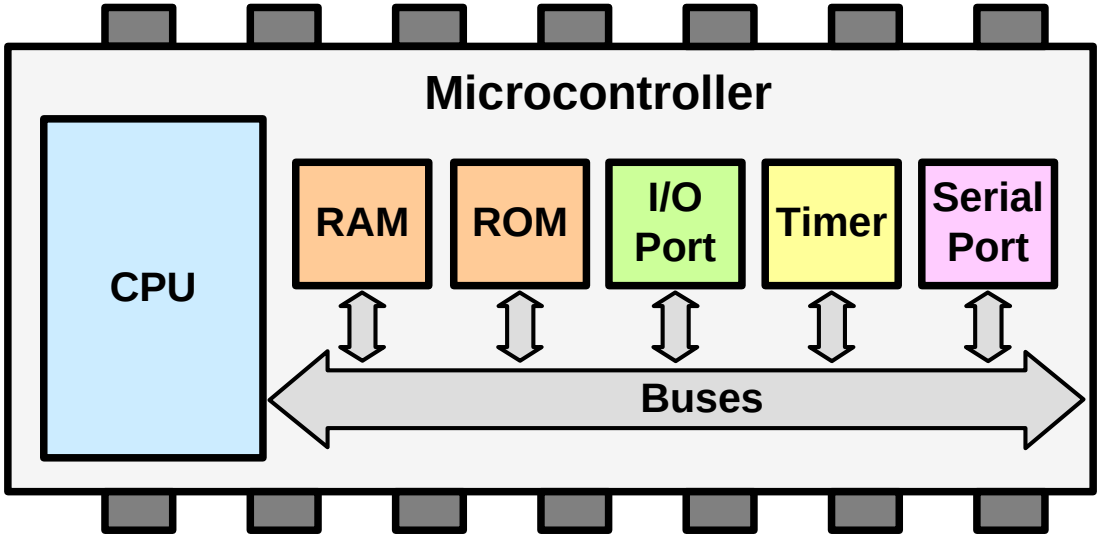
- Microcontrollers vs. Microprocessors
- Most common microcontrollers
- STM32 family

General Purpose Microprocessors vs. Microcontrollers

- *General Purpose Microprocessors*



- *Microcontrollers (MCUs)*



Most common microcontrollers

- 8-bit microcontrollers
  - AVR
  - PIC
  - HCS12
  - 8051

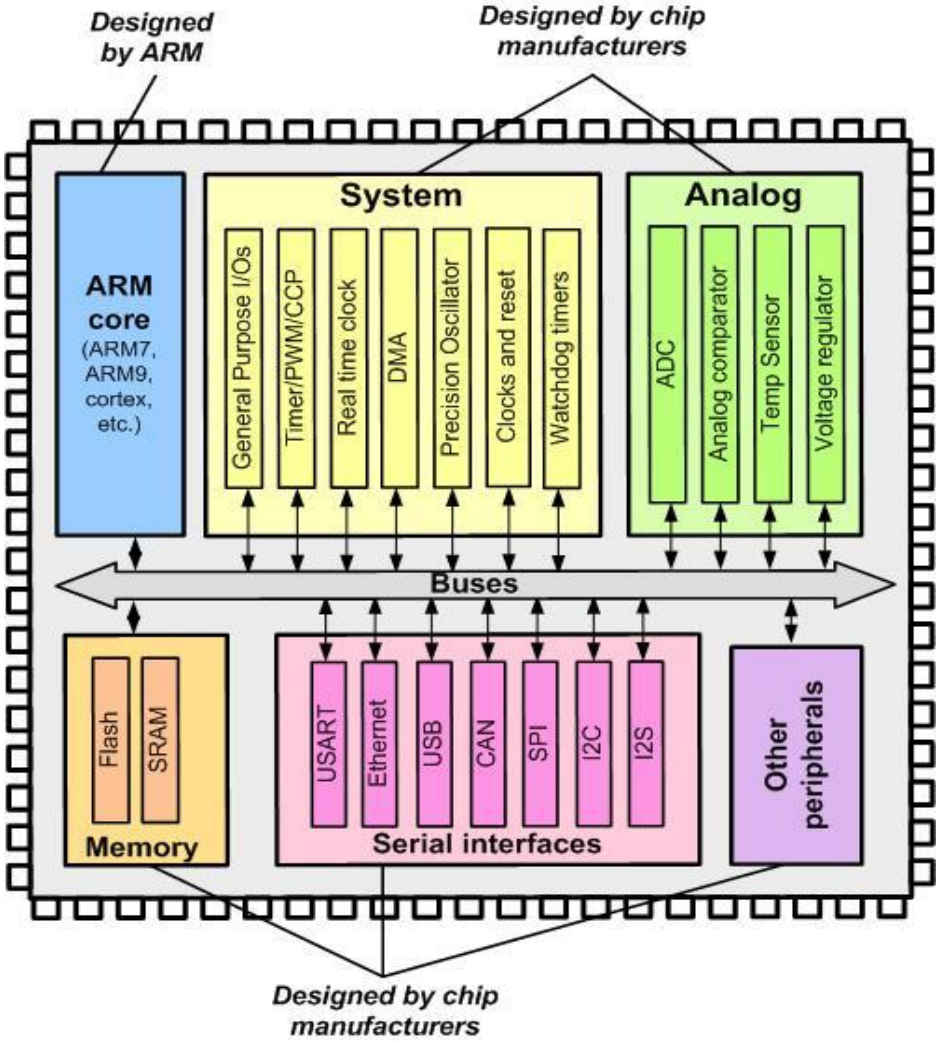
- 32-bit microcontrollers
  - ARM
  - AVR32
  - PIC32

Arm

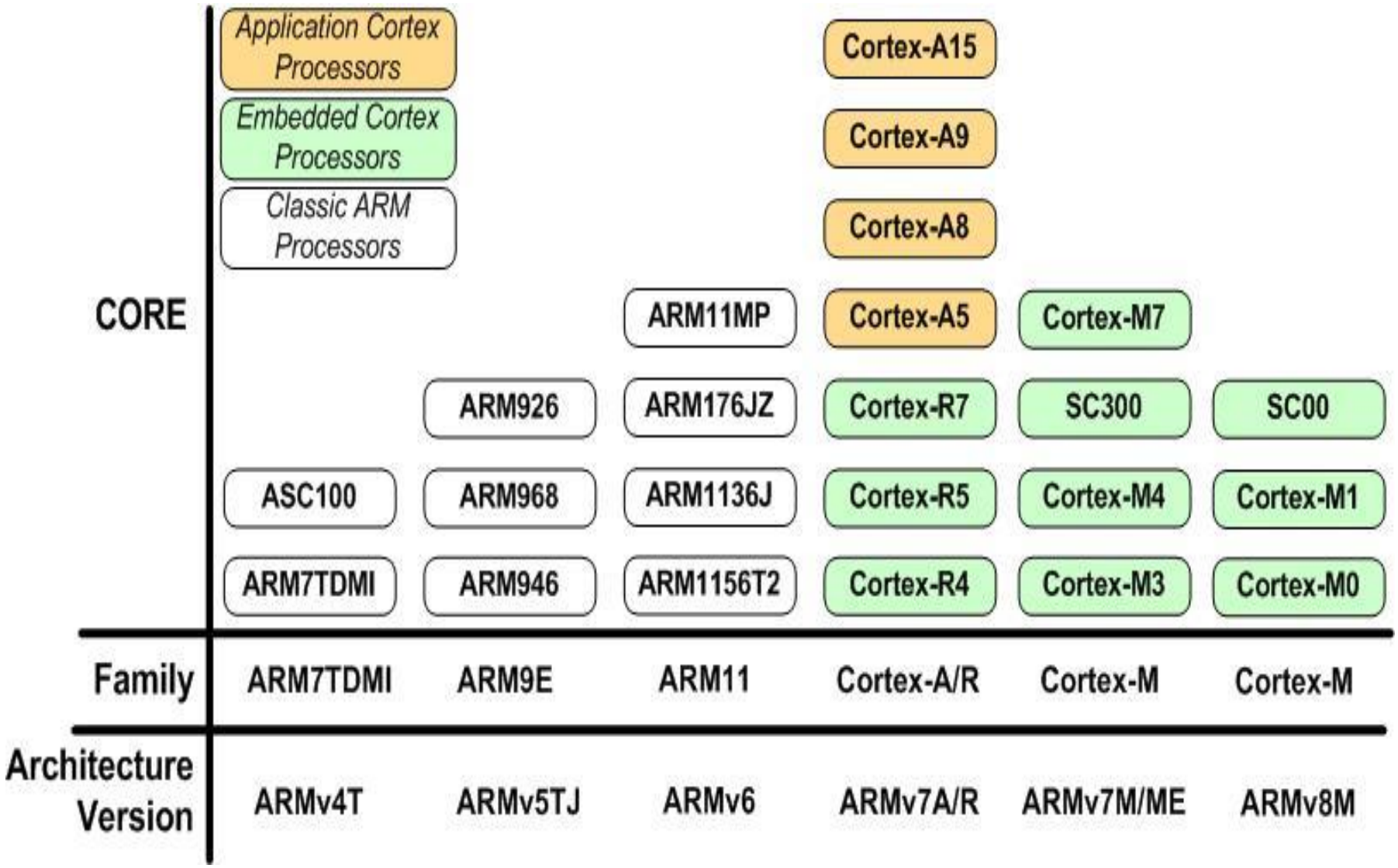
- Arm does not produce chips but designs & sells the license of its of architecture to others.
- More than 200 companies have bought the Arm architecture and provide their Arm chips.

Arm MCUs of Different Companies

- The same CPU and Assembly Language
- Different Peripherals



Arm families and Architectures



STM32 Family

- ST is a multinational company
  - STM8: 8-bit MCUs
  - STM32: 32-bit MCUs based on Arm
    - Low prices
    - Various chips

life.augmented

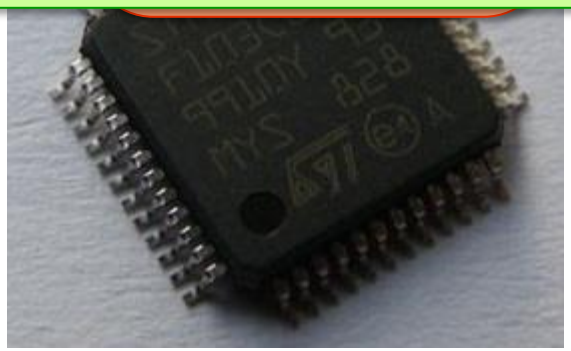
ST32 Naming Conventions

S T M 3 2 F 1 0 3 C 8 T

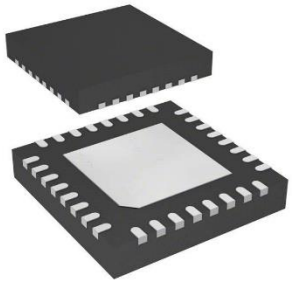
Package
H: BGA (Ball Grid Array)
T:LQFP (Low-profile Quad Flat Pack)
U:QFN (Quad Flat No-leads)
Y:WLCSP

	4	6	8							I
Group	Low density		Mediu							
Flash	16K	32K	64K	128K	256K	384K	512K	768K	1M	1.5M	2M

STM32F429AIH6

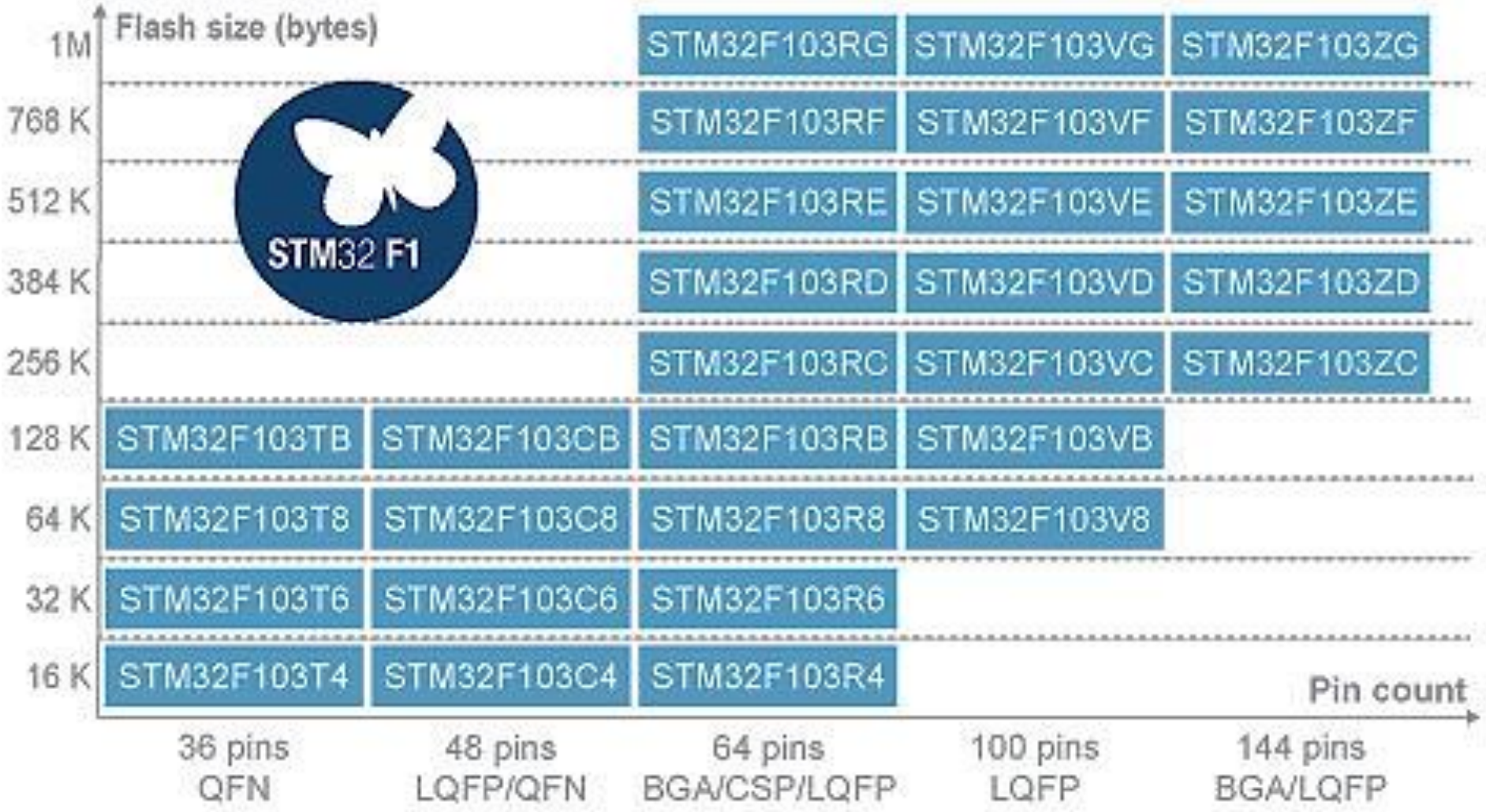


STM32F103C8T6



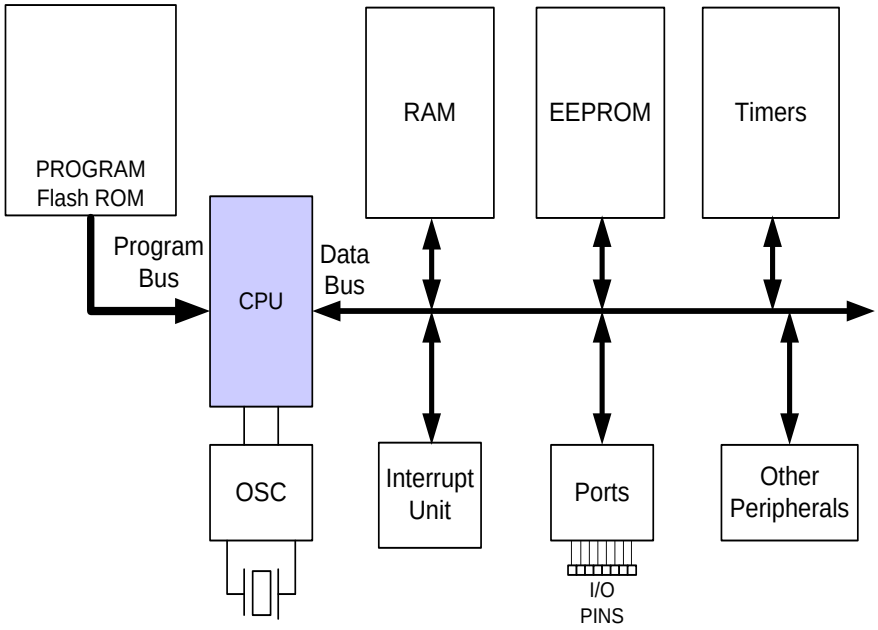
STM32F031K6U6

STM32F103


Flash size (bytes)
1M
768 K
512 K
384 K
256 K
128 K
64 K
32 K
16 K
STM32 F1
STM32F103RG
STM32F103VG
STM32F103ZG
STM32F103RF
STM32F103VF
STM32F103ZF
STM32F103RE
STM32F103VE
STM32F103ZE
STM32F103RD
STM32F103VD
STM32F103ZD
STM32F103RC
STM32F103VC
STM32F103ZC
STM32F103TB
STM32F103CB
STM32F103RB
STM32F103VB
STM32F103T8
STM32F103C8
STM32F103R8
STM32F103V8
STM32F103T6
STM32F103C6
STM32F103R6
STM32F103T4
STM32F103C4
STM32F103R4
Pin count
36 pins QFN
48 pins LQFP/QFN
64 pins BGA/CSP/LQFP
100 pins LQFP
144 pins BGA/LQFP

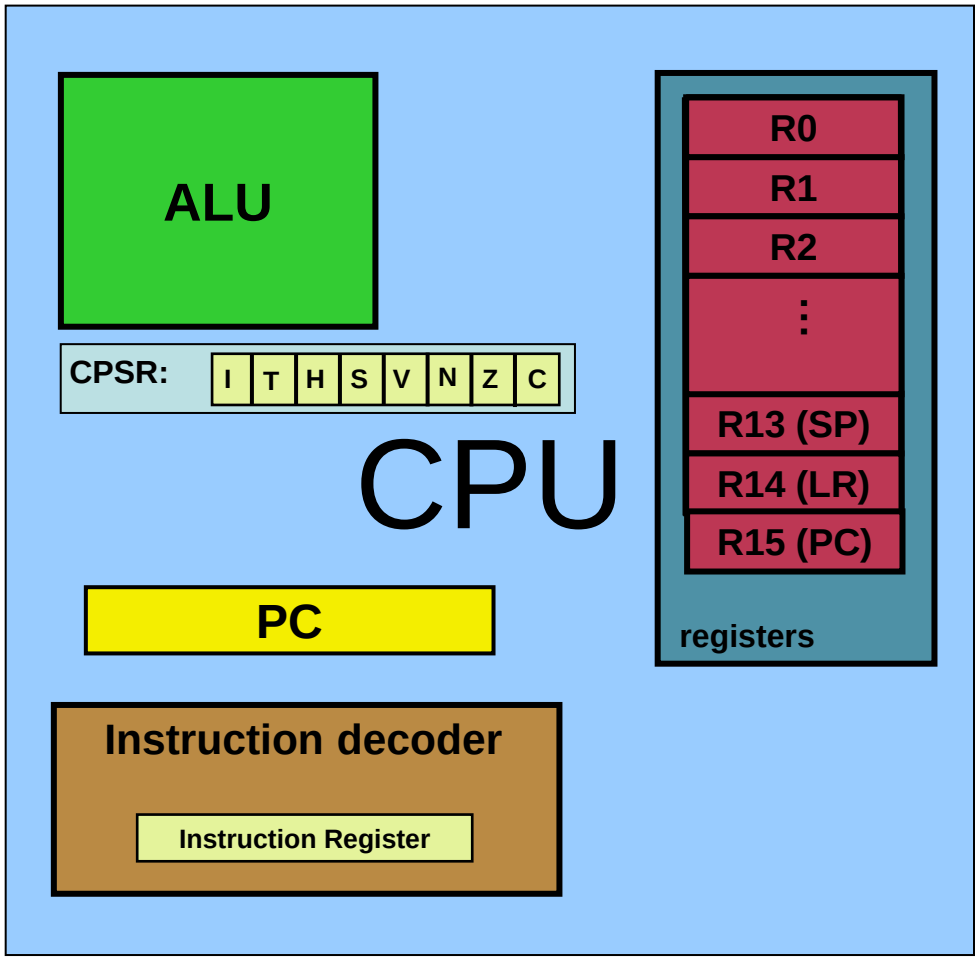
Topics

- ARM's CPU
  - Its architecture
  - Some simple programs
- Data Memory access
- Program memory
- RISC architecture

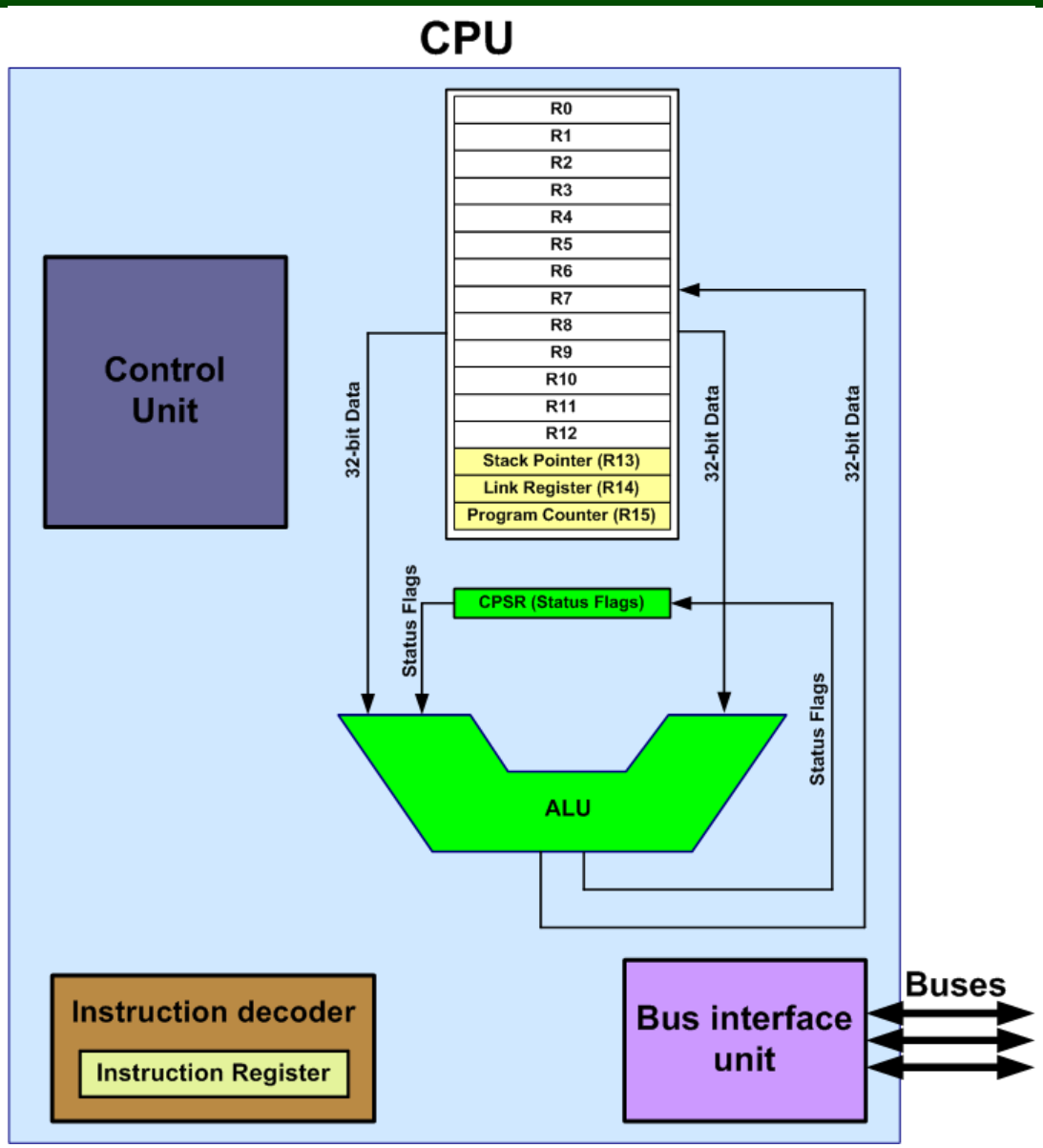


ARM ’s CPU

- ARM ’s CPU
  - ALU
  - 16 General Purpose registers (R0 to R15)
  - PC register (R15)
  - Instruction decoder



CPU


CPU
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
Stack Pointer (R13)
Link Register (R14)
Program Counter (R15)
Control Unit
32-bit Data
32-bit Data
32-bit Data
Status Flags
CPSR (Status Flags)
Status Flags
ALU
Instruction decoder
Instruction Register
Bus interface unit
Buses

Some simple instructions
1. MOV (**MOV**E)

- MOV Rd, #k
  - Rd = k
  - k is an 8-bit value
- Example:
  - MOV R5,#53
    - R5 = 53
  - MOV R9,#0x27
    - R9 = 0x27
  - MOV R3,#2_11101100

- MOV Rd, Rs
  - Rd = Rs
- Example:
  - MOV R5,R2
    - R5 = R2
  - MOV R9,R7
    - R9 = R7

LDR pseudo-instruction (loading 32-bit values)

- LDR Rd, =k
  - Rd = k
  - k is an 32-bit value
- Example:
  - LDR R5,=5543
    - R5 = 5543
  - LDR R9,=0x123456
    - R9 = 0x123456
  - LDR R4,=2_1011011001011001

Some simple instructions

2. Arithmetic calculation

- Or
  –
- Ex
  –
  –
  –

Instruction		Description
ADD	**Rd, Rn,Op2 ***	ADD Rn to Op2 and place the result in Rd
ADC	**Rd, Rn,Op2**	ADD Rn to Op2 with Carry and place the result in Rd
AND	**Rd, Rn,Op2**	AND Rn with Op2 and place the result in Rd
BIC	**Rd, Rn,Op2**	AND Rn with NOT of Op2 and place the result in Rd
CMP	**Rn,Op2**	Compare Rn with Op2 and set the status bits of CPSR**
CMN	**Rn,Op2**	Compare Rn with negative of Op2 and set the status bits
EOR	**Rd, Rn,Op2**	Exclusive OR Rn with Op2 and place the result in Rd
MVN	**Rd,Op2**	Store the negative of Op2 in Rd
MOV	**Rd,Op2**	Move (Copy) Op2 to Rd
ORR	**Rd, Rn,Op2**	OR Rn with Op2 and place the result in Rd
RSB	**Rd, Rn,Op2**	Subtract Rn from Op2 and place the result in Rd
RSC	**Rd, Rn,Op2**	Subtract Rn from Op2 with carry and place the result in Rd
SBC	**Rd, Rn,Op2**	Subtract Op2 from Rn with carry and place the result in Rd
SUB	**Rd, Rn,Op2**	Subtract Op2 from Rn and place the result in Rd
TEQ	**Rn,Op2**	Exclusive-OR Rn with Op2 and set the status bits of CPSR
TST	**Rn,Op2**	AND Rn with Op2 and set the status bits of CPSR

Op2 can be an immediate 8-bit value #K which can be 0–255 in decimal, (00–FF in hex). Op2 can also be a register Rm. Rd, Rn and Rm are any of the general purpose registers

*** CPSR is discussed later in this chapter***

A simple program

- Write a program that calculates 19 + 95

```
MOV R6, #19     ;R6 = 19
MOV R2, #95     ;R2 = 95
ADD R6, R6, R2  ;R6 = R6 + R2
```

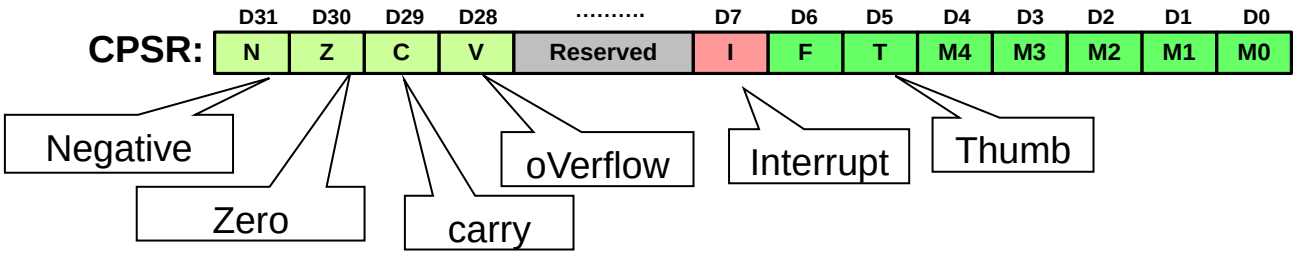
A simple program

- Write a program that calculates 19 + 95 - 5

```
MOV     R1, #19         ;R6 = 19
MOV     R2, #95         ;R2 = 95
MOV     R3, #5          ;R21 = 5
ADD     R6, R1,R2       ;R6 = R1 + R2
SUB     R6, R6,R3       ;R6 = R6 - R3
```

```
MOV     R1, #19         ;R6 = 19
MOV     R2, #95         ;R2 = 95
ADD     R6, R1,R2       ;R6 = R1 + R2
MOV     R2, #5          ;R21 = 5
SUB     R6, R6,R2       ;R6 = R6 - R2
```

Status Register (CPSR)



Example: Show the status of the Z flag after the subtraction of 0x23 from 0xA5 in the following instructions:

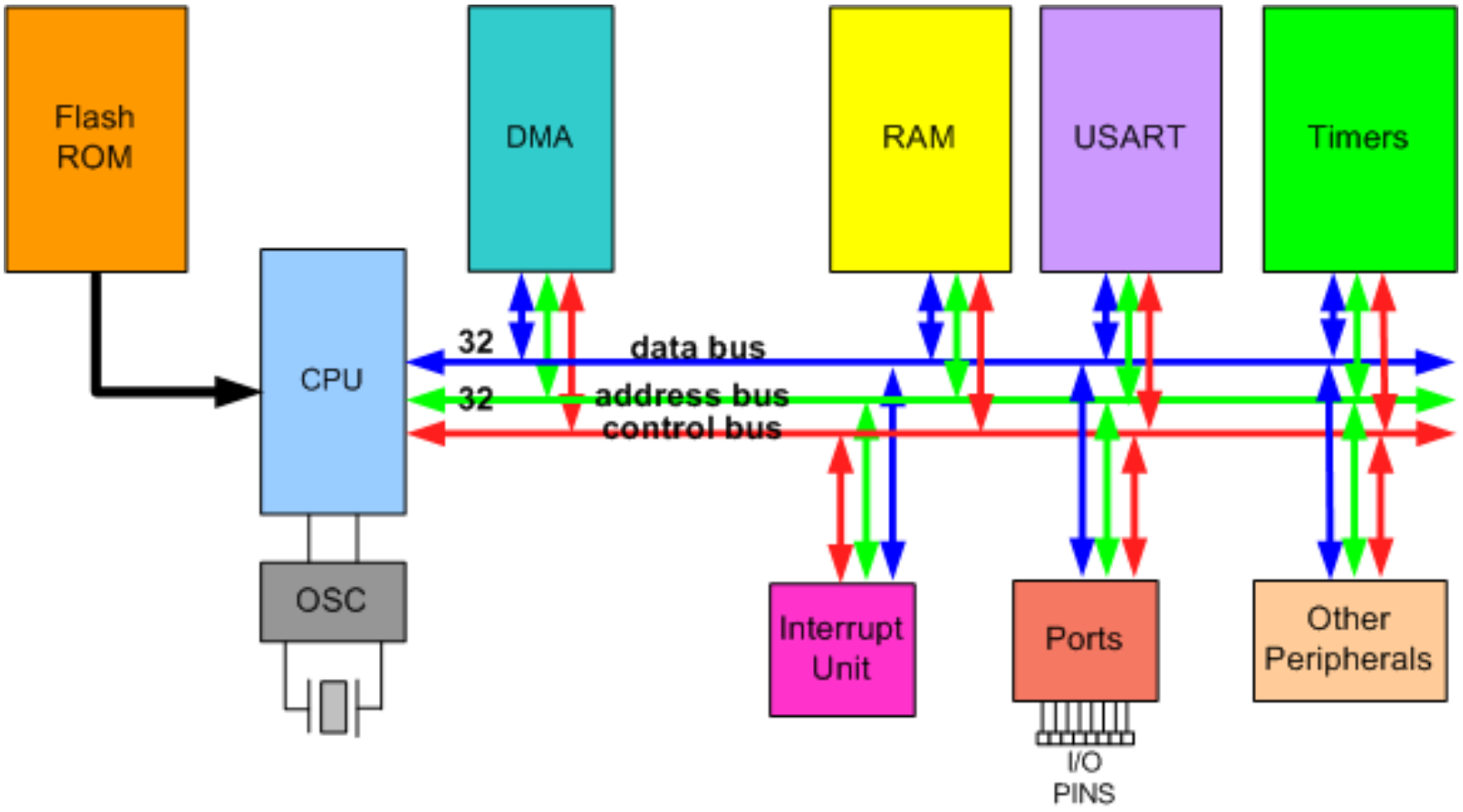
```
LDR     R0,=0xA5
LDR     R1,=0x23
SUBS    R0,R0,R1          ;subtract R1 from R0
```

Solution:

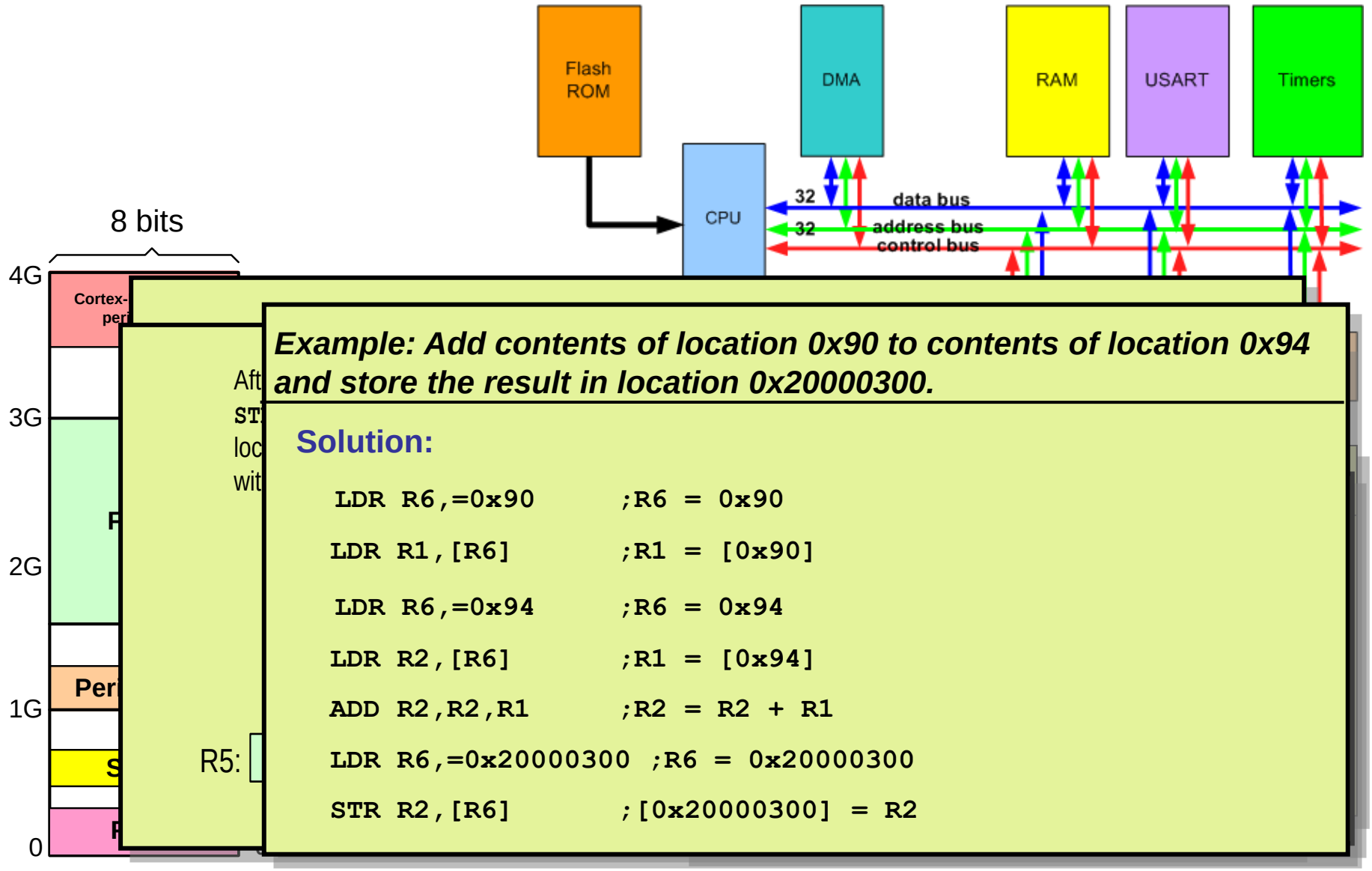
	0xA5	1010 0101	
-	0x23	0010 0011	
	0x82	1000 0010	R0 = 0x82

Z = 0 because the R20 has a value other than 0 after the subtraction.
C = 1 because R1 is not bigger than R0 and there is no borrow from D32 bit.

Harvard in ARM9 and Cortex



Memory Map in STM32F103



Example: Add contents of location 0x90 to contents of location 0x94 and store the result in location 0x20000300.

Solution:

```
LDR R6,=0x90        ;R6 = 0x90
LDR R1,[R6]         ;R1 = [0x90]
LDR R6,=0x94        ;R6 = 0x94
LDR R2,[R6]         ;R1 = [0x94]
ADD R2,R2,R1        ;R2 = R2 + R1
LDR R6,=0x20000300  ;R6 = 0x20000300
STR R2,[R6]         ;[0x20000300] = R2
```

LDRB, LDRH, STRB, STRH

Data Size	Bits	Load instruction used	Store instruction used
Byte	8	LDRB	STRB
Half-word	16	LDRH	STRH
Word	32	LDR	STR

LDR Rd,[Rs]

LDRB Rd,[Rs]
LDRH Rd,[Rs]

STR Rs,[Rd]

STRB Rs,[Rd]
STRH Rs,[Rd]

Assume that R5=0x40000200, and R1 = 0x41526374.
After running the following instruction:

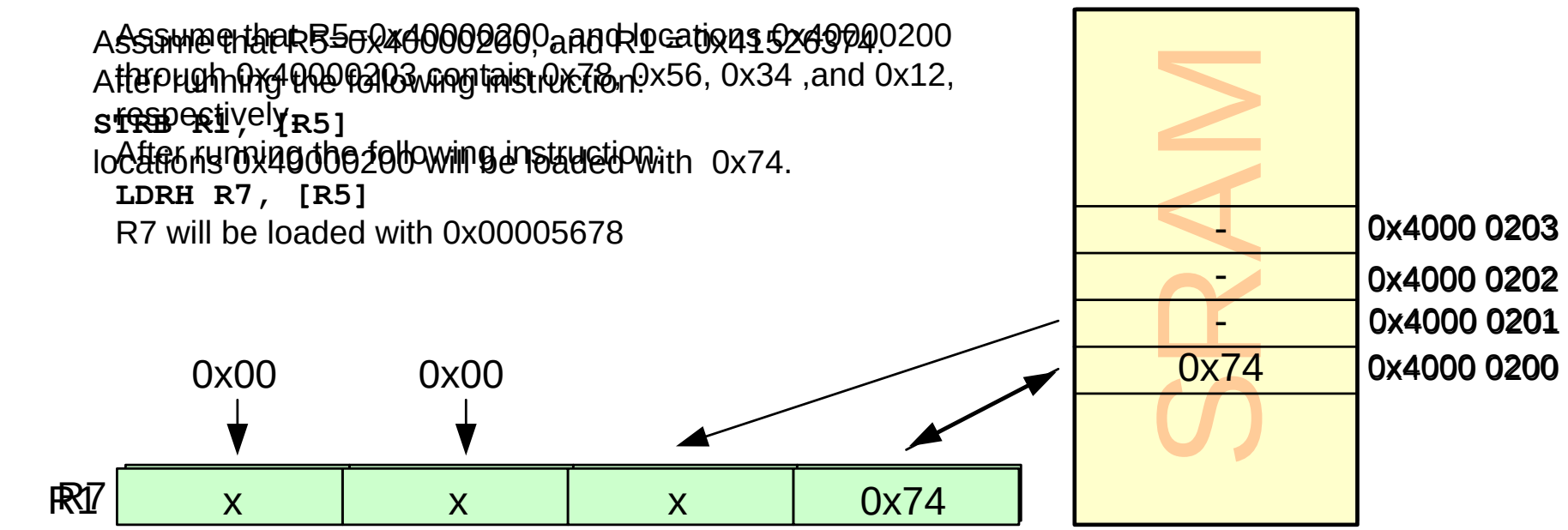
```
STRB R1, [R5]
```

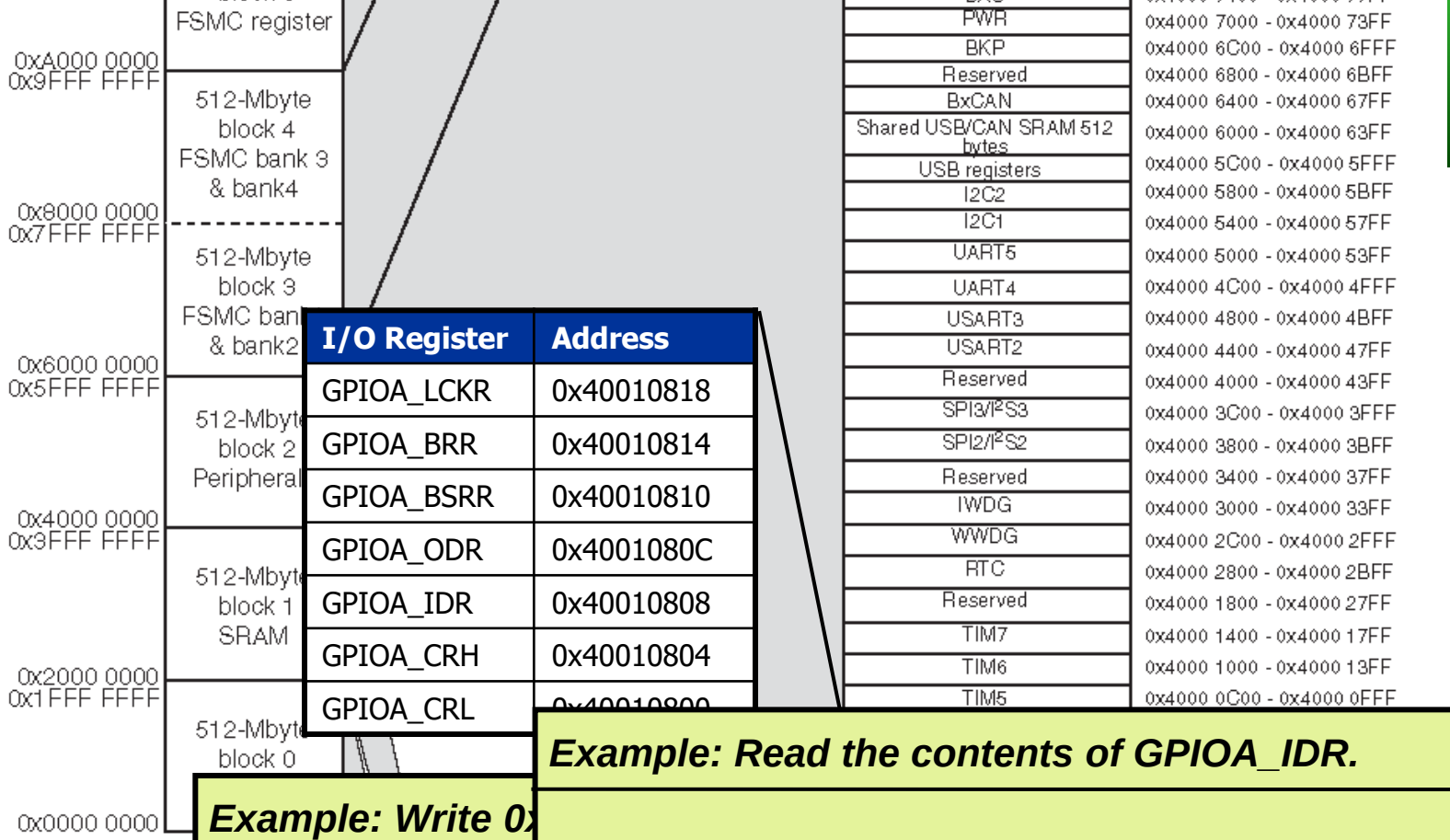
locations 0x40000200 will be loaded with 0x74.

Assume that R5=0x40000200, and locations 0x40000200 through 0x40000203 contain 0x78, 0x56, 0x34 ,and 0x12, respectively.
After running the following instruction:

```
LDRH R7, [R5]
```

R7 will be loaded with 0x00005678





I/O Register	Address
GPIOA_LCKR	0x40010818
GPIOA_BRR	0x40010814
GPIOA_BSRR	0x40010810
GPIOA_ODR	0x4001080C
GPIOA_IDR	0x40010808
GPIOA_CRH	0x40010804
GPIOA_CRL	

Example: Write 0x

Solution:

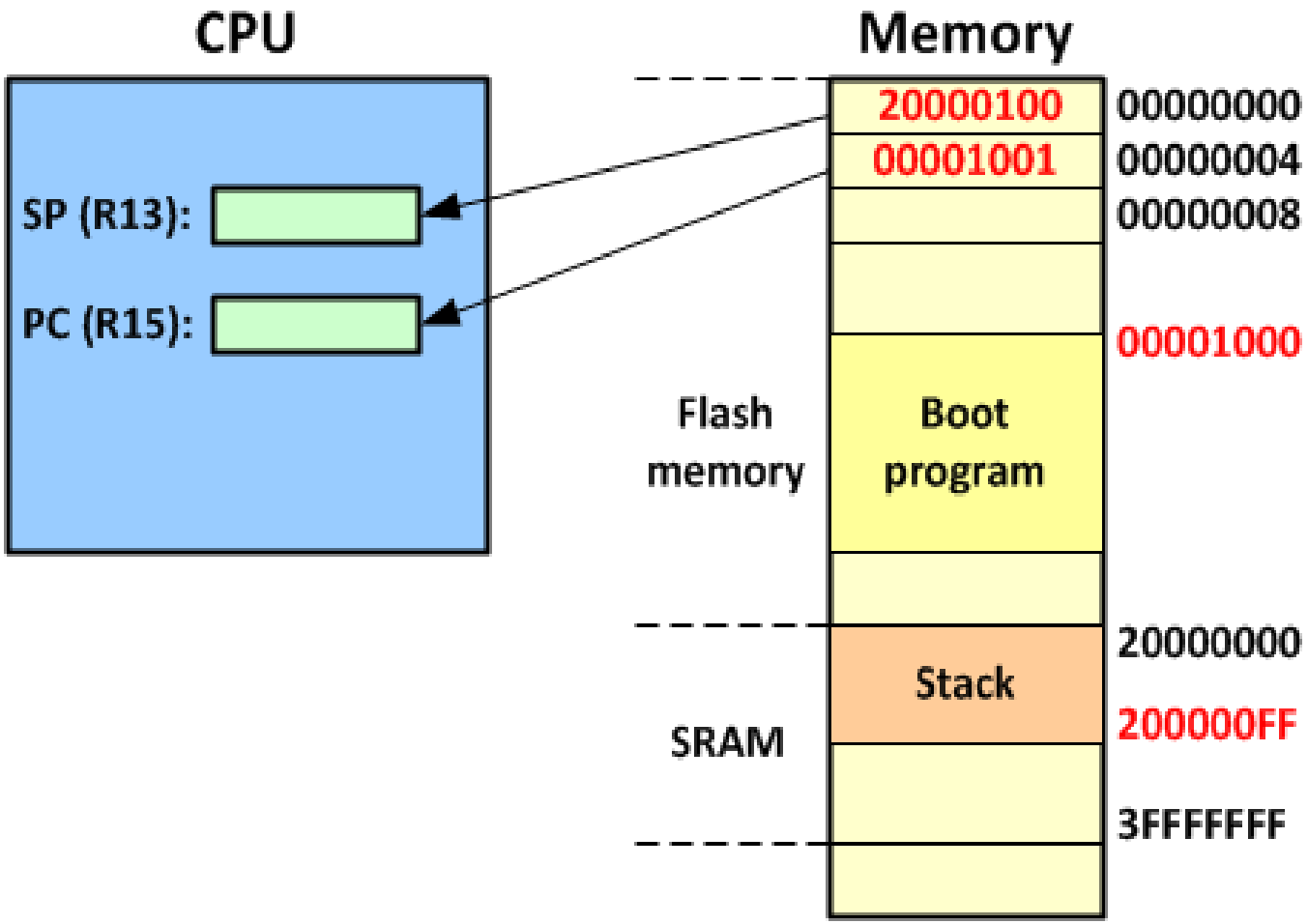
```
LDR R2,=0x5
LDR R1,=0x4
STR R2,[R1]
```

Example: Read the contents of GPIOA_IDR.

Solution:

```
LDR R1,=0x40010808 ;R1= 0x40010808
LDR R2,[R1]        ;R2 = [0x4001080C]
```

Power up in Cortex-M



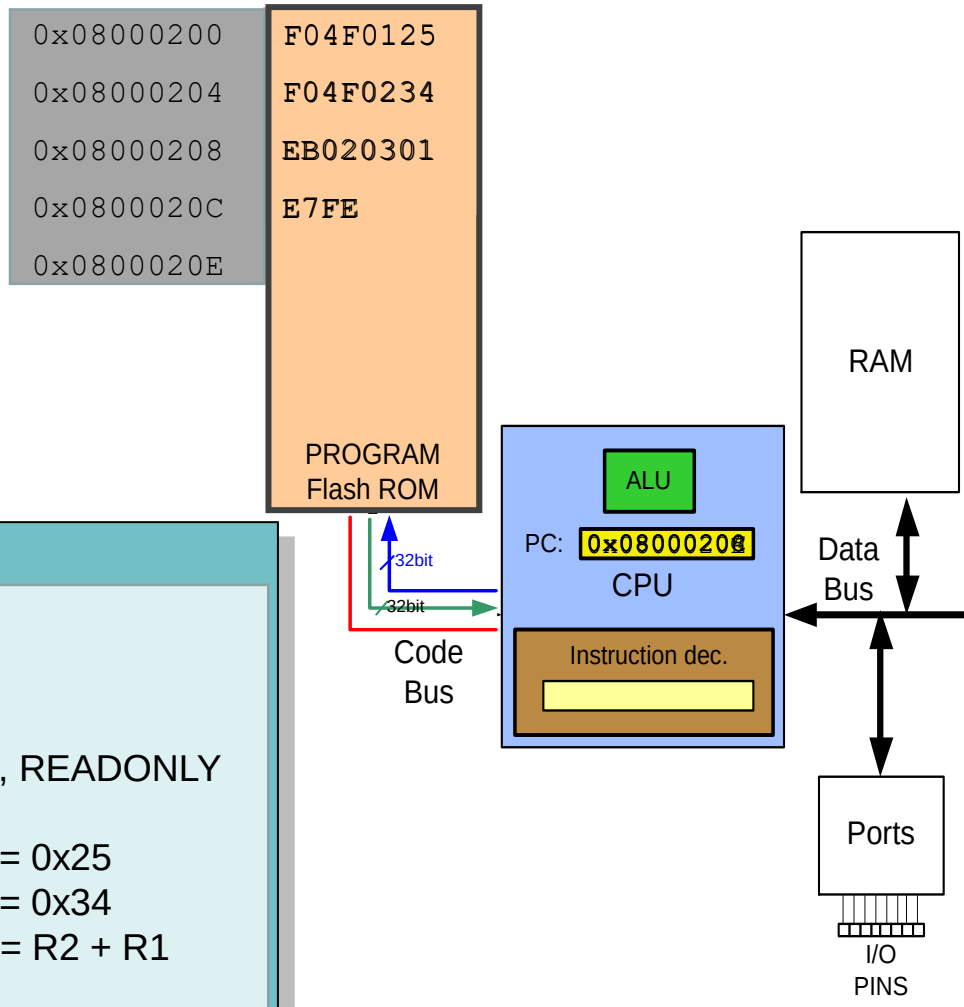
Startup and main files

Startup_stm32f10x.s

```
124   .section .isr_vector,"a",%progbits
125   .type g_pfnVectors, %object
126   .size g_pfnVectors, .-g_pfnVectors
127
128
129 g_pfnVectors:
130
131   .word _estack
132   .word Reset_Handler
133   .word NMI_Handler
134   .word HardFault_Handler
```

```
35 /* Entry Point */
36 ENTRY(Reset_Handler)
37
38 /* Highest address of the user mode stack */
39 _estack = ORIGIN(RAM) + LENGTH(RAM); /* end of "RAM" Ram type memory */
40
41 _Min_Heap_Size = 0x200; /* required amount of heap */
42 _Min_Stack_Size = 0x400; /* required amount of stack */
43
44 /* Memories definition */
45 MEMORY
46 {
47   RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 20K
48   FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 64K
49 }
```

Flash memory and PC register



main.lst

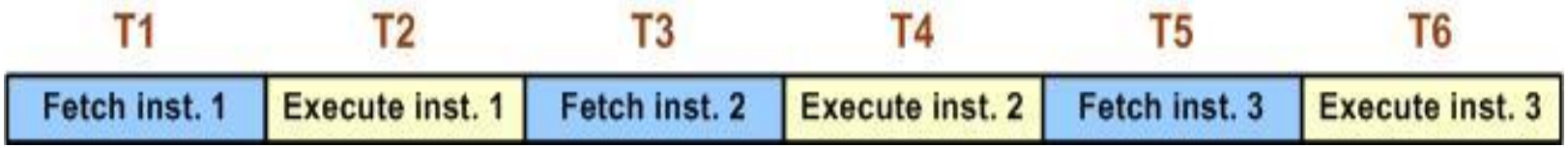
```
Line Offset   Machine      Instruction
 1  00000000              ; The program adds some data
 2  00000000                EXPORT    __main
 3  00000000                AREA      PROG_2_4, CODE, READONLY
 4  00000000              __main
 5  00000000  F04F 0125     MOV     R1, #0x25   ; R1 = 0x25
 6  00000004  F04F 0234     MOV     R2, #0x34   ; R2 = 0x34
 7  00000008  EB02 0301     ADD     R3, R2, R1  ; R3 = R2 + R1
 8  0000000C
 9  0000000C  E7FE    HERE  B       HERE        ; stay here forever
10  0000000E                END
```

How to speed up the CPU

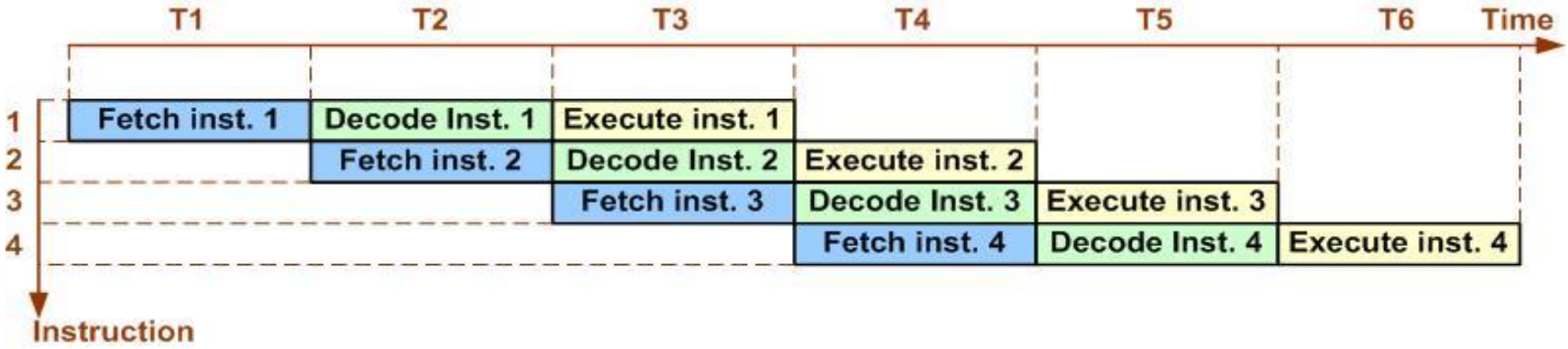
- Increase the clock frequency
  - More frequency ➔ More power consumption & more heat
  - Limitations
- Change the architecture
  - Pipelining
  - Harvard
  - RISC

Pipeline

- Non-pipeline
  - Just fetches, decodes, or executes in a given time

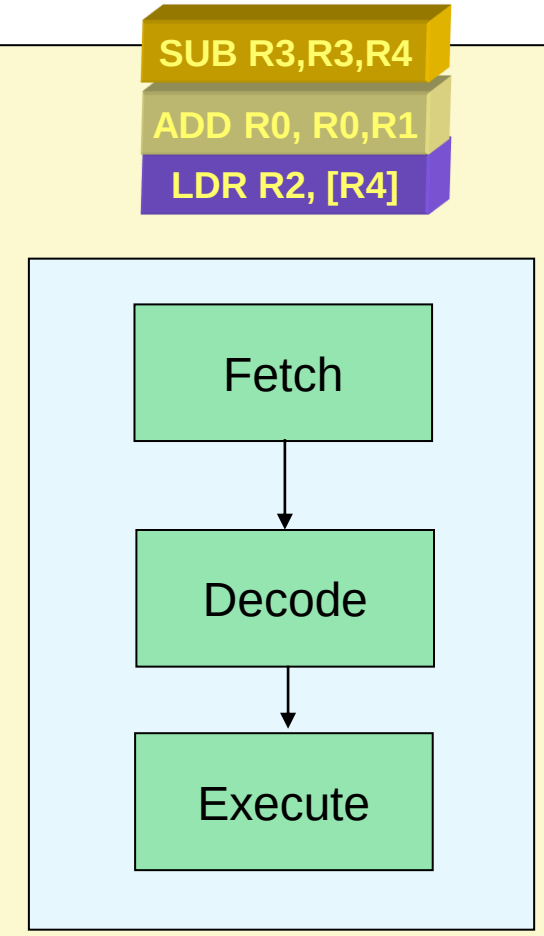


- Pipeline



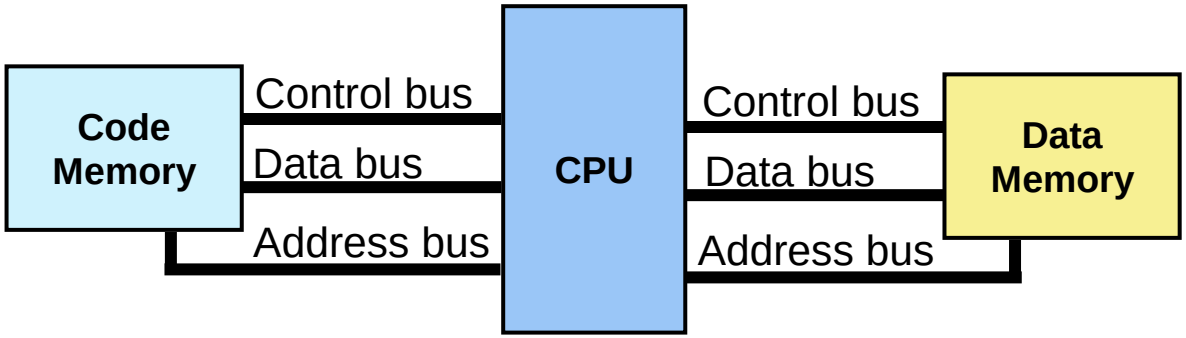
Pipeline (Cont.)

```
LDR R2, [R4]    ; R2 = [R4]
ADD R0,R0,R1    ; R20 = R20 + R21
SUB R3,R3,R4
```



Harvard Architecture

- separate buses for opcodes and operands
  - Advantage: opcodes and operands can go in and out of the CPU together.
  - Disadvantage: Using Harvard architecture in motherboards leads to more cost in general purpose computers.



Changing the architecture
RISC vs. CISC

- CISC (Complex Instruction Set Computer)
  - Put as many instruction as you can into the CPU
- RISC (Reduced Instruction Set Computer)
  - Reduce the number of instructions, and use your facilities in a more proper way.

RISC architecture

- Feature 1 (fixed instruction size)
  - RISC processors have a fixed instruction size. It makes the task of instruction decoder easier.
    - In ARM the instructions are 4 bytes.
    - In Thumb2 the instructions are either 2 or 4 bytes.
  - In CISC processors instructions have different lengths
    - E.g. in 8051
      - CLR C ; a 1-byte instruction
      - ADD A, #20H ; a 2-byte instruction
      - LJMP HERE ; a 3-byte instruction

RISC architecture

- Feature 2: reduce the number of instructions
  - Pros: Reduces the number of used transistors
  - Cons:
    - Can make the assembly programming more difficult
    - Can lead to using more memory

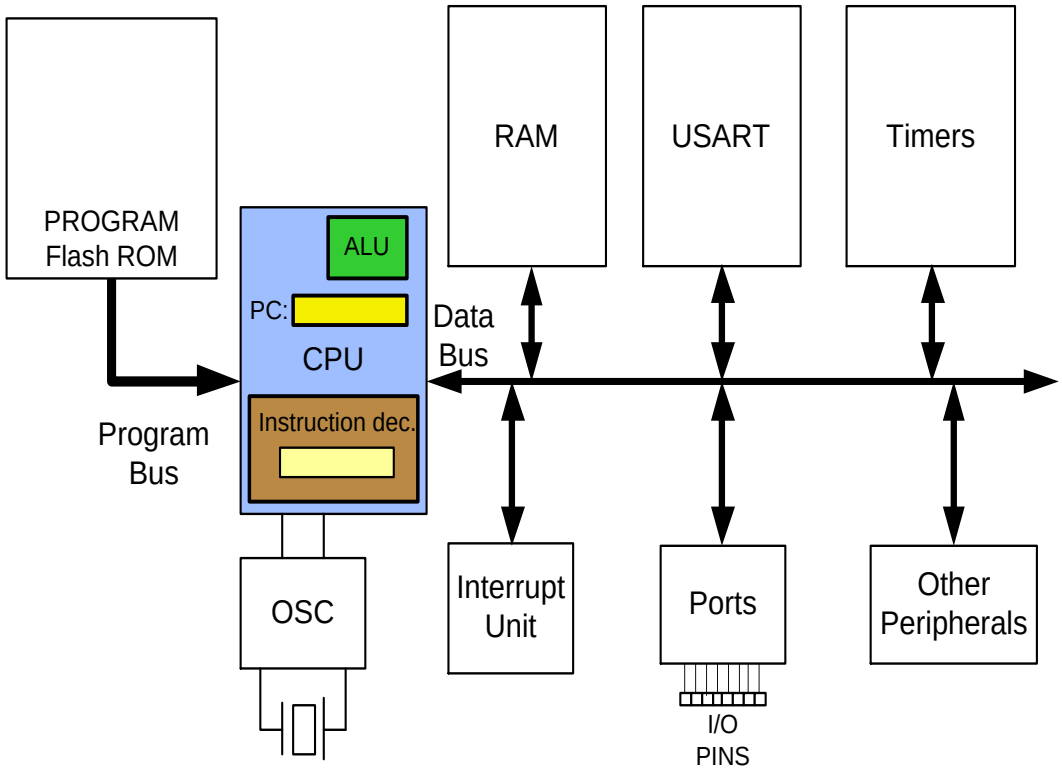
RISC architecture

- Feature 3: limit the addressing mode
  - Advantage
    - hardwiring
  - Disadvantage
    - Can make the assembly programming more difficult

RISC architecture

- Feature 4: Load/Store

```
LDR R8,=0x20
LDR R0,[R8]
LDR R8,=0x220
LDR R1,[R8]
ADD R0, R0,R1
LDR R8,=0x230
STR R0,[R8]
```



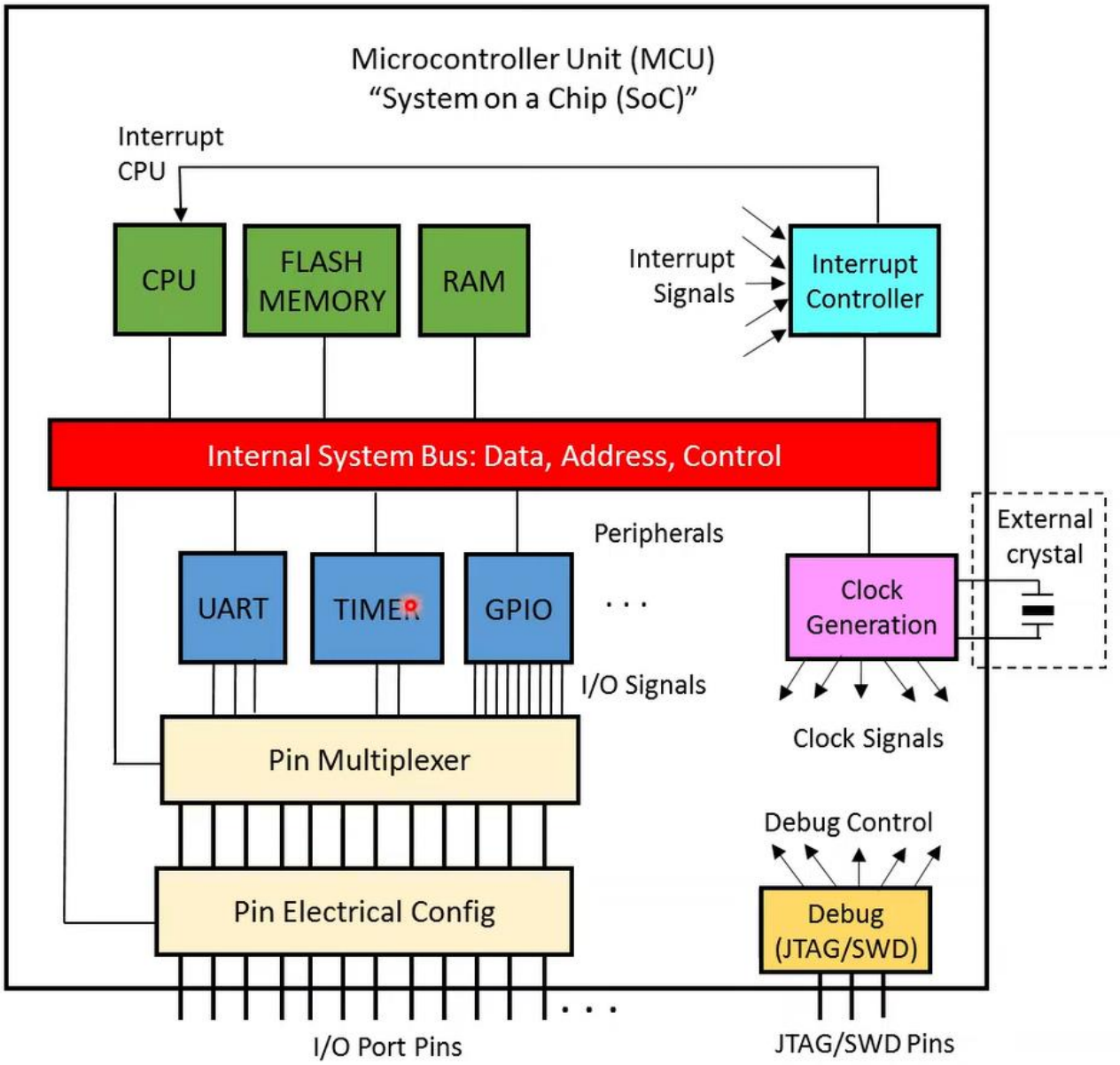
RISC architecture

- Feature 5: more than 95% of instructions are executed in 1 machine cycle

RISC architecture

- Feature 6
  - RISC processors have at least 32 registers. Decreases the need for stack and memory usages.
    - In ARM there are 16 general purpose registers (R0 to R15)

SOC block diagram



SOC block diagram