

Computer Organization & Cortex-M Architecture

Microcontroller Application and Development
Sorayut Glomglome

π

Outline

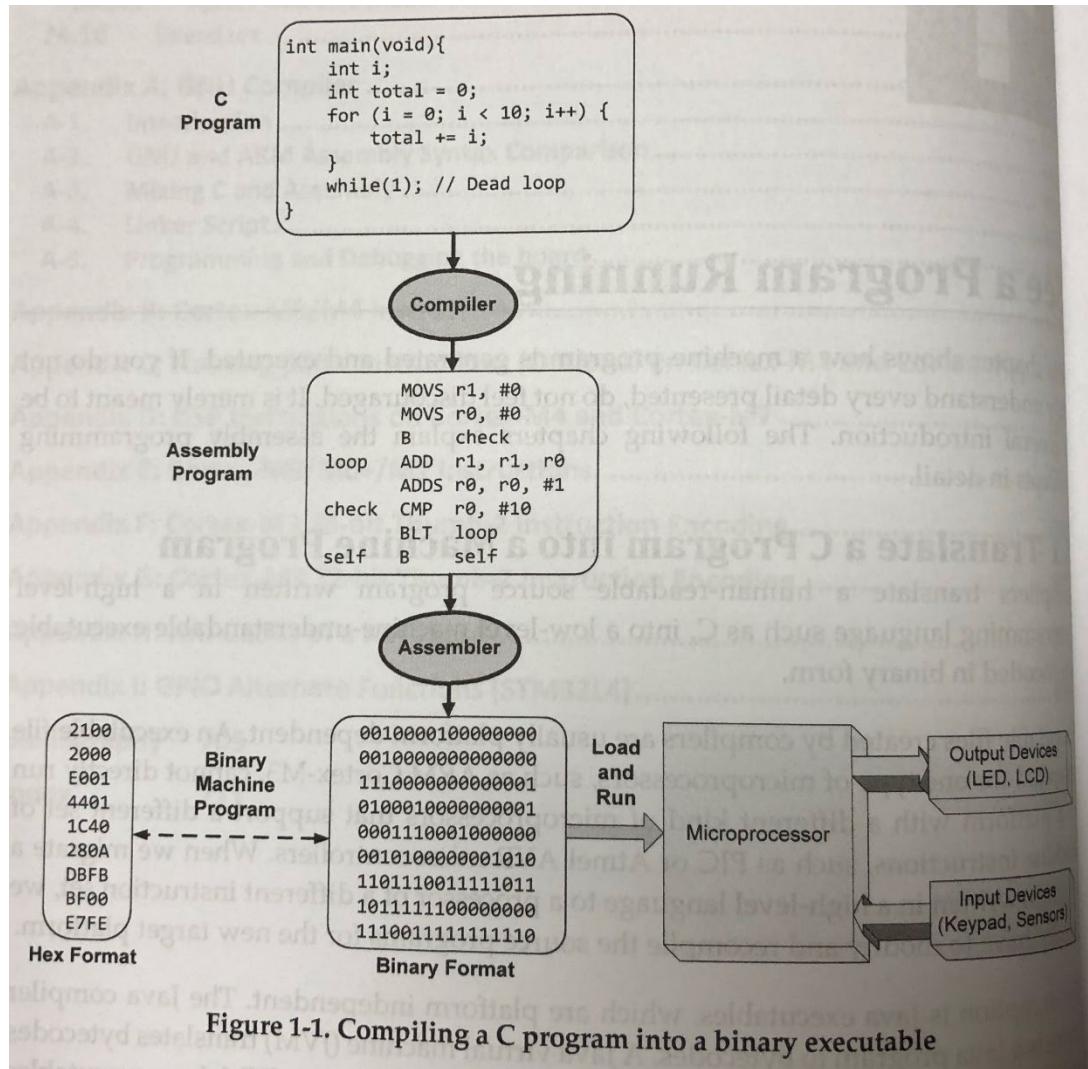
1. Computer Organization
2. Cortex M Architecture
3. Processor Registers
4. Memory Technology
5. Assembly Language

Learning Outcome

- Understand Computer Organization
- Understand Cortex M Architecture
- Understand Basic of ARM Cortex Assembly

การแปลภาษาสูงเป็นภาษา แอสเซมบลีและภาษาเครื่อง

- ภาษาระดับสูง (High-level language)
 - มีความใกล้เคียงกับโจทย์หรือปัญหา
 - โปรแกรมเมอร์สามารถเขียนโปรแกรมได้ง่ายและสร้างสรรค์ผลงานได้มากกว่า
- ภาษาแอสเซมบลี (Assembly language)
 - คำสั่งภาษาเครื่องในรูปของตัวหนังสือ
- ภาษาเครื่อง (Machine Language)
 - ชุดของเลขฐานสองที่เรียงตัวต่อกันจำนวนเท่าของ 8 บิต เช่น 8, 16, 32, 64 บิต
 - แต่ละบิตมีค่าเป็น 0 หรือ 1 ก็ได้ โดย 0 หรือ 1 นี้จะอยู่ในรูปของระดับโวตเตจ



Program Build Tools

Program Build Tools

Figure 5.1 gives an overview of the build tools and the files they process to create the final executable file. Programs are built of modules that are translated through a series of formats and then combined into an executable file. The file `my_module.c` contains the module's code in the C language format. This is compiled to create an assembly language module in the file `my_module.s`, which is then assembled to create an object module in the file `my_module.o`. This module is linked with other modules (also in object format) to create the executable file.

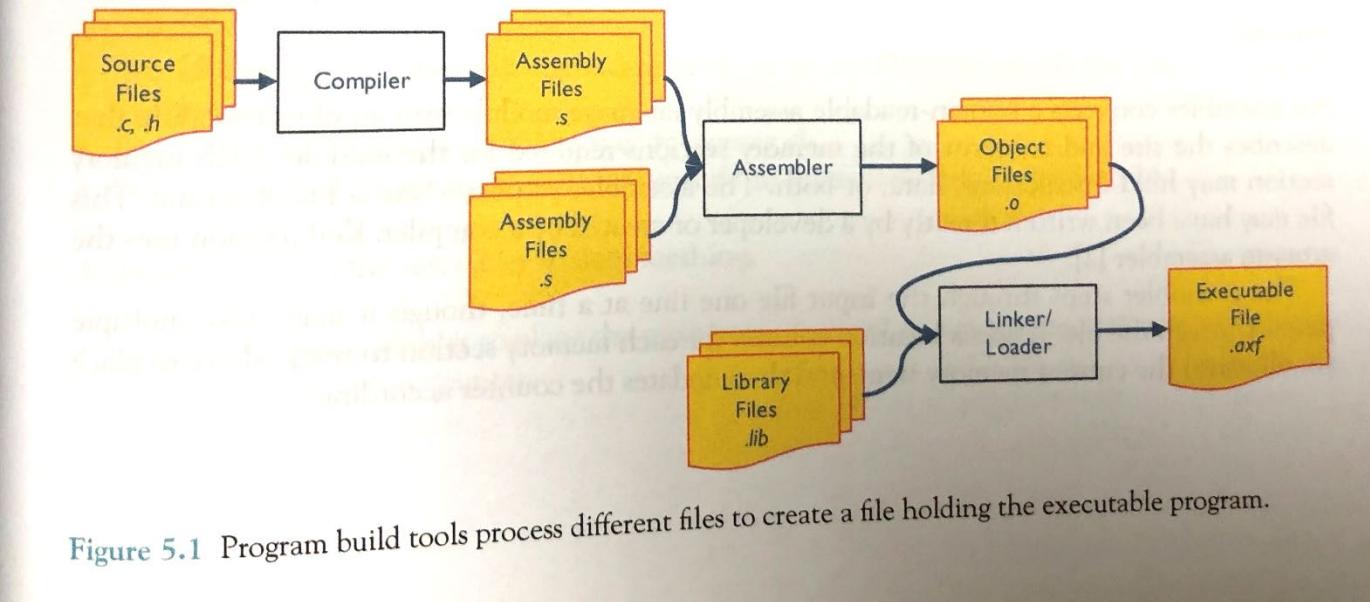
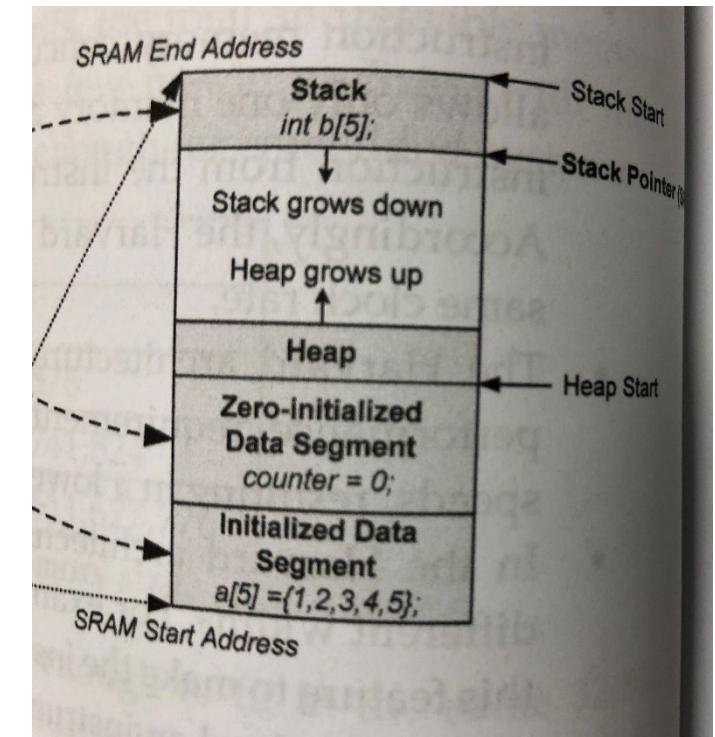
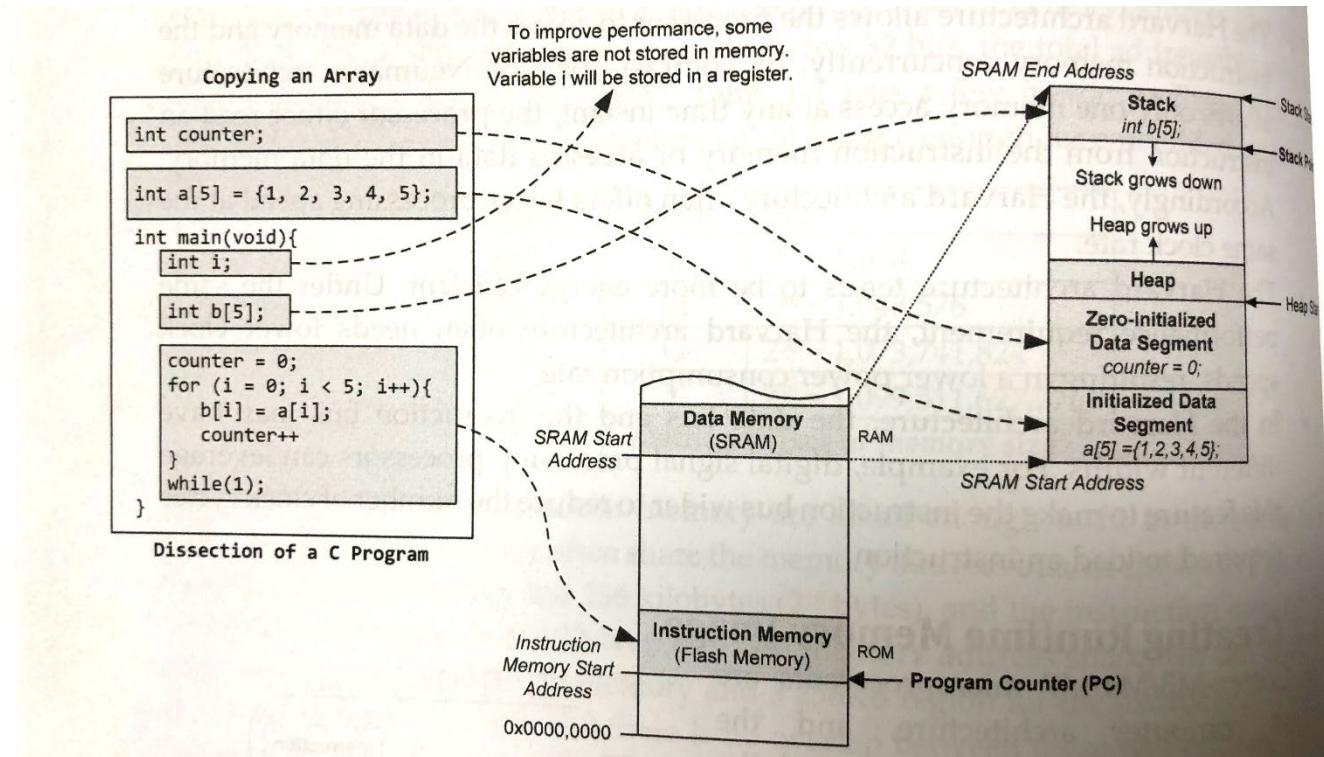
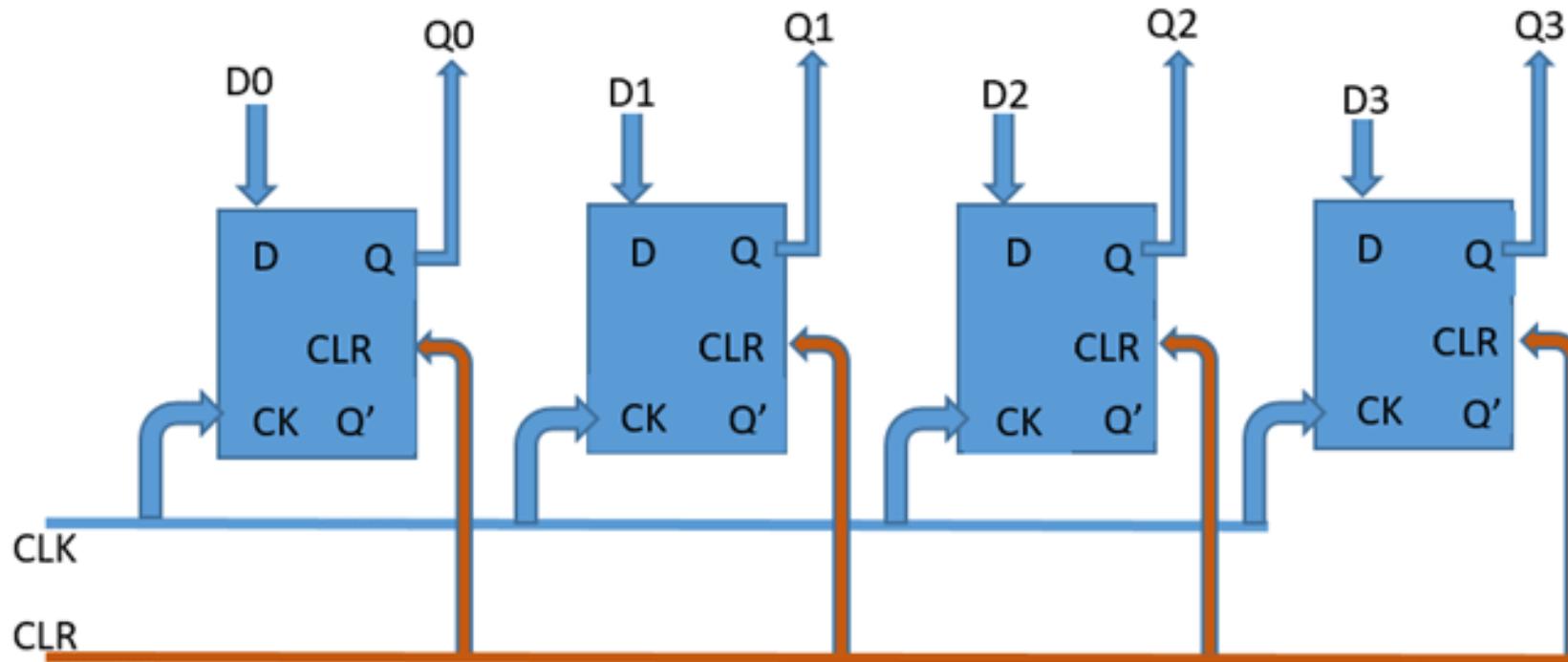


Figure 5.1 Program build tools process different files to create a file holding the executable program.

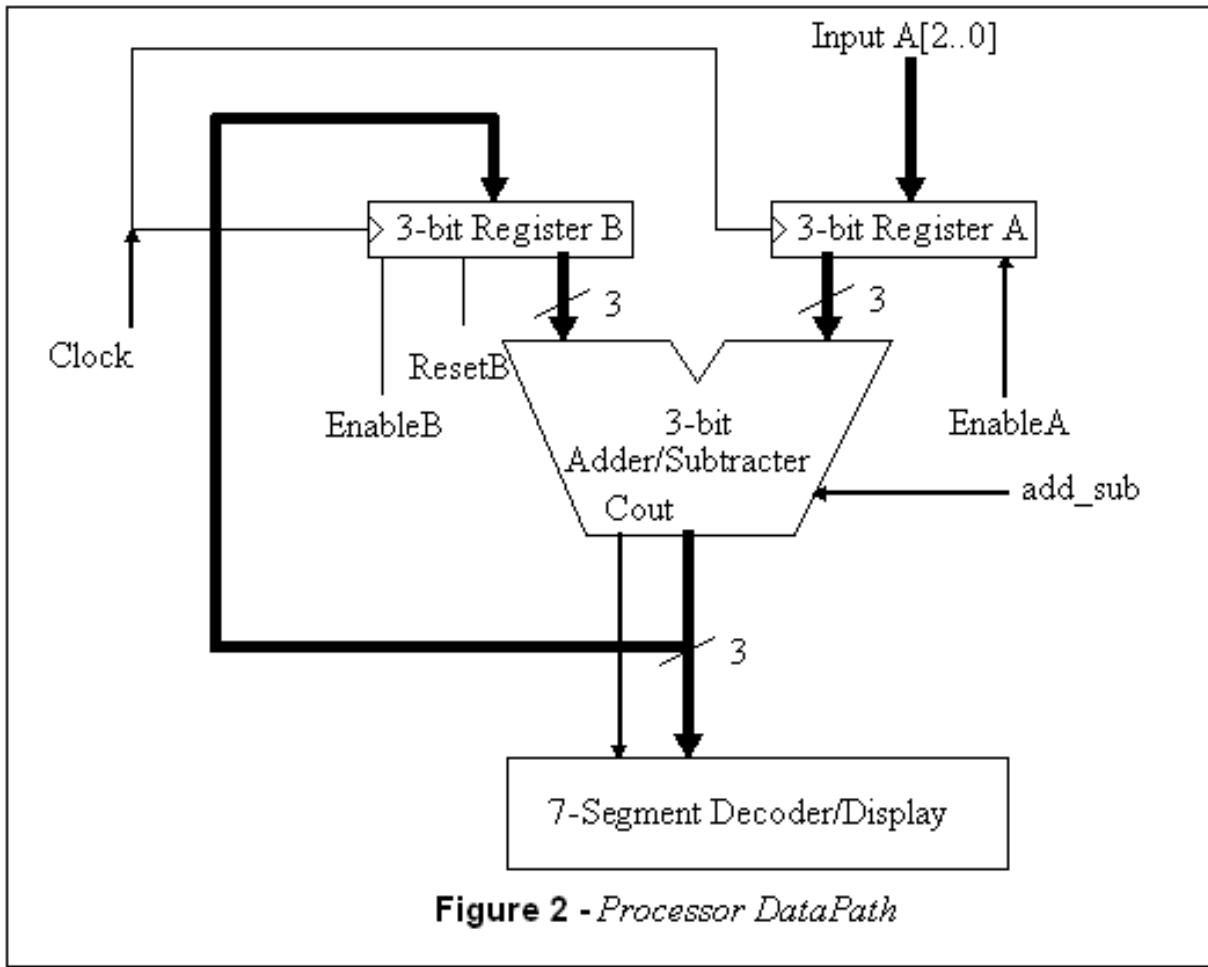
Harvard Processor Program Loading



4 Bit Register from D Flip-Flop



ALU with Registers



Assembly Code Example 1

Example 3.6: Write code that reads from variable **N** multiplies by 5, adds 25, and stores the result in variable **M**. Both variables are 32-bit.

Solution: First, we perform a 32-bit read, bringing **N** into Register **R1**. Second we multiply by 5 and add 10, and lastly we store the result into **M**. Since the value gets larger, overflow could occur. This solution ignores the overflow error.

| | |
|--|--|
| <pre>LDR R3, =N ; R3 = &N (R3 points to N) LDR R1, [R3] ; R1 = N MOV R0, #5 ; R0 = 5 MUL R1, R0, R1 ; R1 = 5*N MOV R0, #25 ; R0 = 25 ADD R0, R0, R1 ; R0 = 25+5*N LDR R2, =M ; R2 = &M (R2 points to M) STR R0, [R2] ; M = 25+5*N</pre> | <pre>// C implementation M = 5*N+25;</pre> |
|--|--|

Program 3.8. Example code showing a 32-bit multiply and addition.

Assembly Code Example 2

Example 3.7: Write code to convert a variable N ranging from 0 to 1023 into a variable M , which ranges from 0 to 3000. Essentially compute $M = 2.93255 * N$. Both variables are 32-bit.

Solution: First, we perform a 32-bit read, bringing N into Register R1. Second we multiply by 3000 and divide by 1023, and lastly we store the result into M . Since the input range is bounded ($3000 * 1023 < 2^{32}$) no overflow error can occur.

| | |
|---|---|
| <pre>LDR R3, =N ; R3 = &N (R3 points to N) LDR R1, [R3] ; R1 = N MOV R0, #3000 ; R0 = 3000 MUL R1, R0, R1 ; 3000*N (0 to 3069000) MOV R0, #1023 ; R0 = 1023 UDIV R0, R1, R0 ; R0 = R1/R0 = 3000*N/1023 LDR R2, =M ; R2 = &M (R2 points to M) STR R0, [R2] ; M = (3000*N)/1023</pre> | <pre>// C implementation M = (3000*N)/1023;</pre> |
|---|---|

Assembly Code Example 3

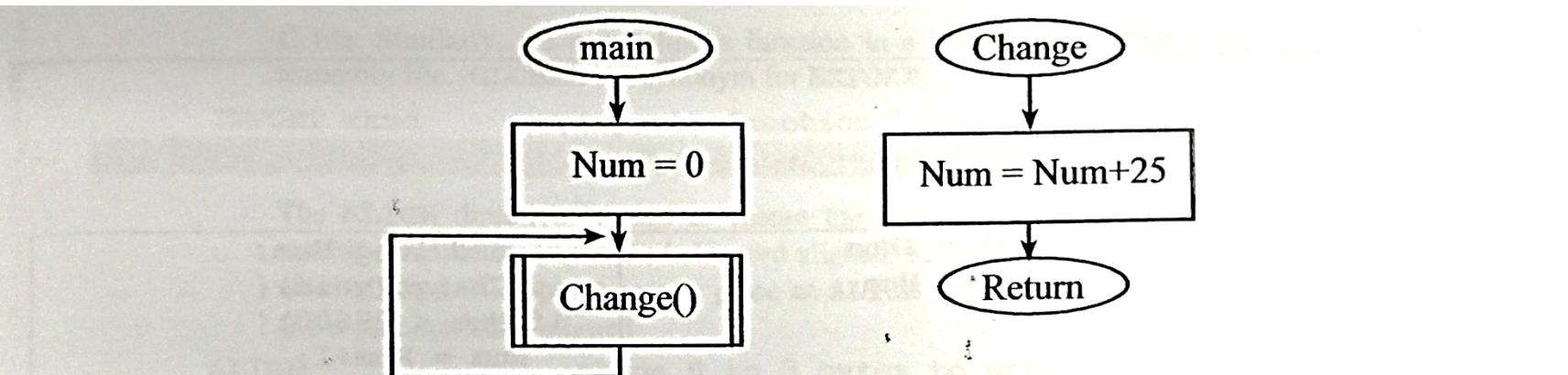


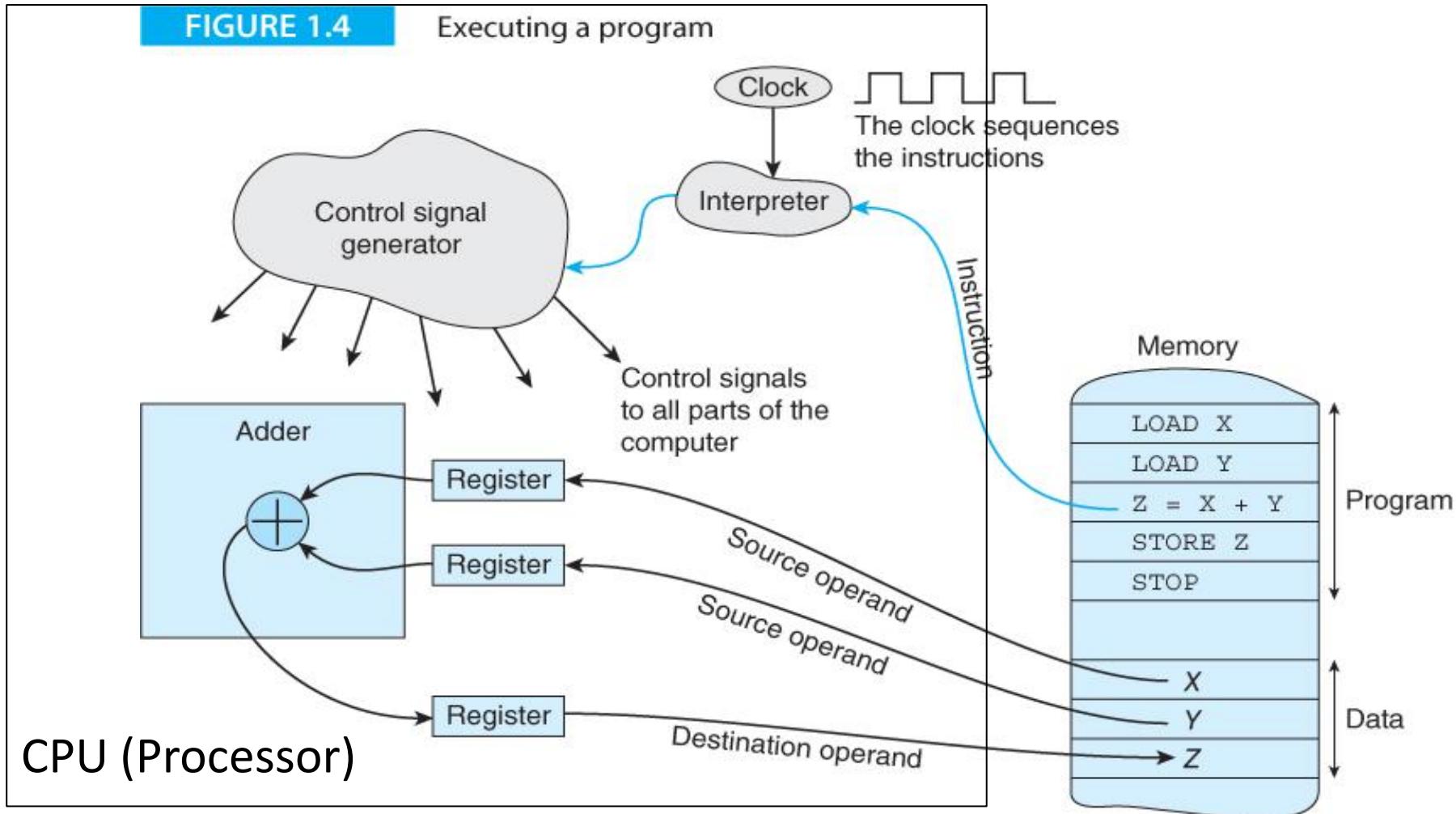
Figure 3.29. A flowchart of a simple function that generates a pseudo random number.

In C, input parameters, if any, are passed in R0–R3. If there are more than 4 input parameters, they are pushed on the stack. The output parameter, if needed, is returned in R0.

| | | |
|--------|--------------------------------|---------------------|
| Change | LDR R1,=Num ; 5) R1 = &Num | unsigned long Num; |
| | LDR R0,[R1] ; 6) R0 = Num | void Change(void) { |
| | ADD R0,R0,#25 ; 7) R0 = Num+25 | Num = Num+25; |
| | STR R0,[R1] ; 8) Num = Num+25 | } |
| | BX LR ; 9) return | void main(void) { |
| main | LDR R1,=Num ; 1) R1 = &Num | Num = 0; |
| | MOV R0,#0 ; 2) R0 = 0 | while(1){ |
| | STR R0,[R1] ; 3) Num = 0 | Change(); |
| loop | BL Change ; 4) function call | } |
| | B loop ; 10) repeat | } |

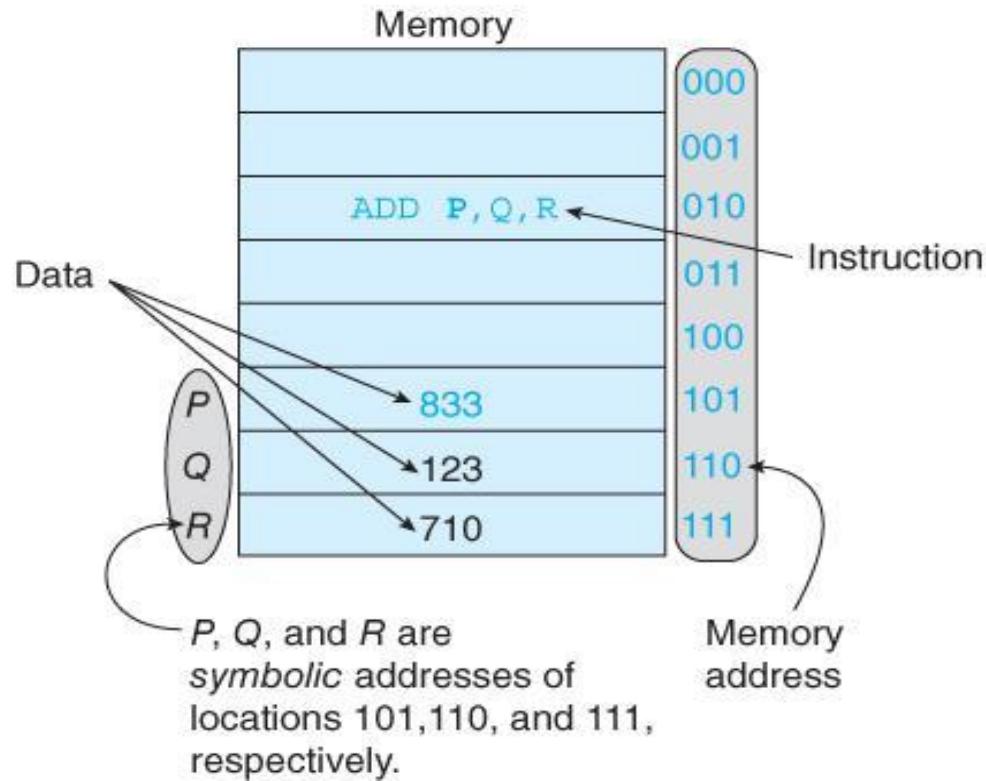
Program 3.9. Assembly and C versions that define a simple function. The 1-2-...-10 show the execution sequence.

How software orders CPU (Processor) to run



Memory System stores Software/Data

FIGURE 1.13 Relationship between instruction and operands



© Cengage Learning 2014

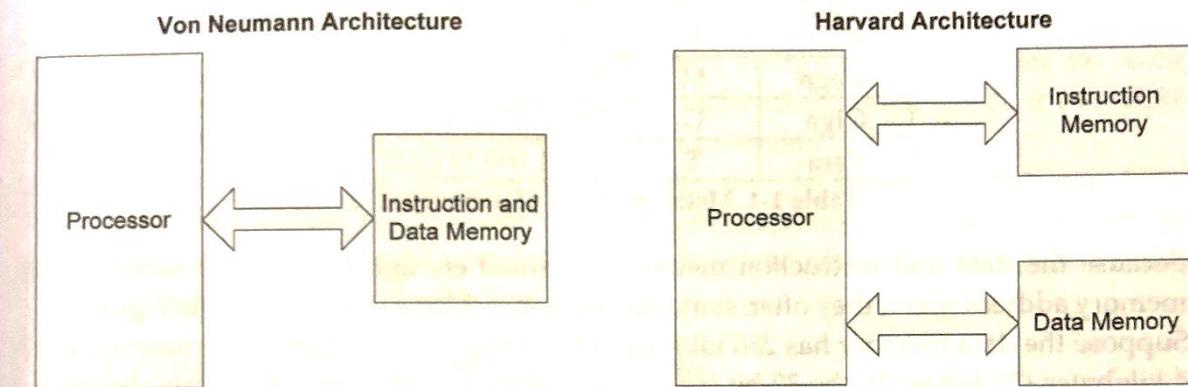


Figure 1-3. Two types of computer architecture. In the Von Neumann architecture, data and instructions are stored in the same memory. In the Harvard architecture, data and instructions are stored in two physically separate memory devices.

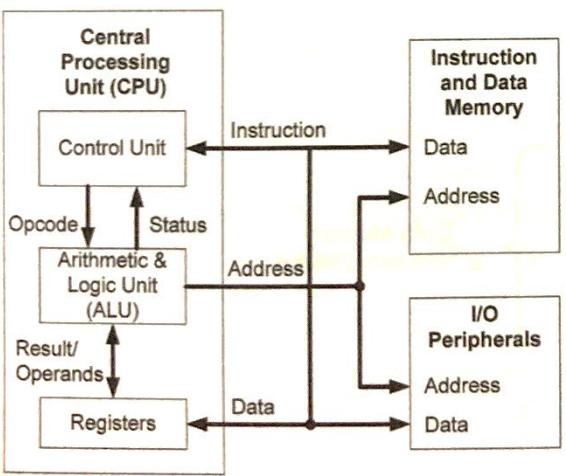


Figure 1-4. Von Neumann computer architecture. Instructions and data share the memory device. It has only one set of data bus and address bus shared by the instruction memory and the data memory.

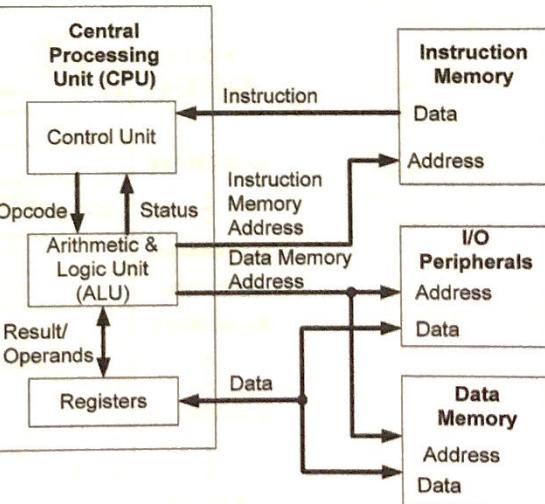
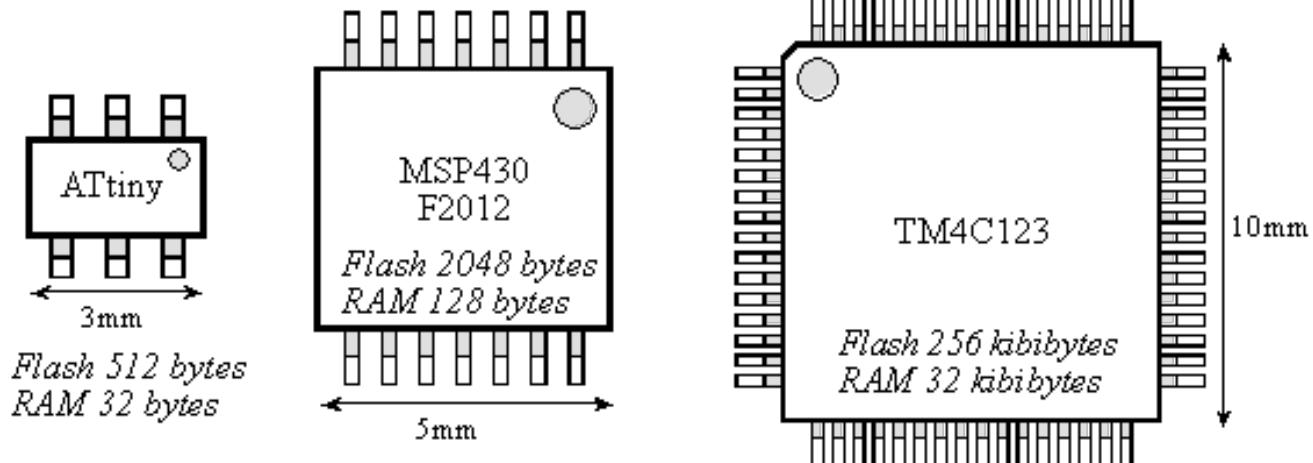


Figure 1-5. Harvard computer architecture. Instructions and data are stored in different memory devices. It has a dedicated set of data bus and address bus for the instruction memory and the data memory.

Von Neumann VS Harvard Architecture

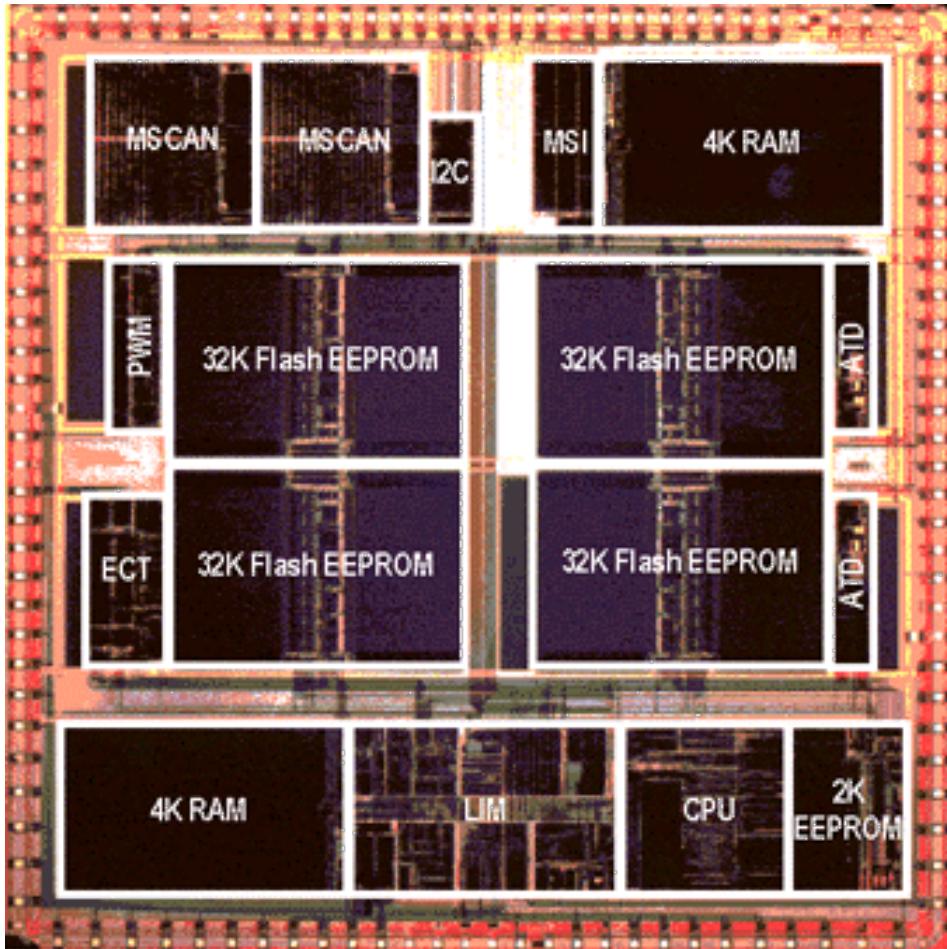
Microcontroller

- A complete computer on a single chip
 - Processor
 - Memory
 - I/O



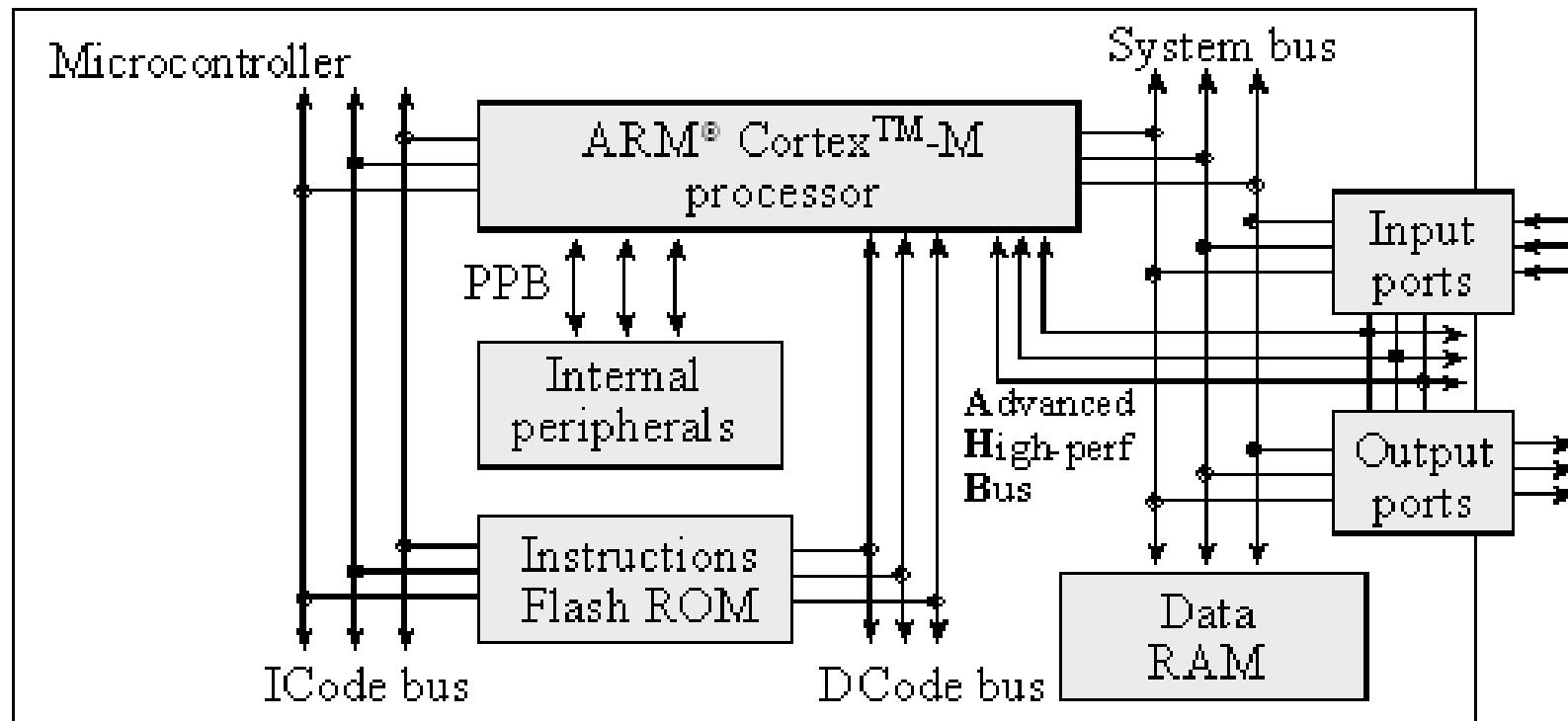
A Photo of MCU Die

- 128K Flash EEPROM
 - 4 x 32K
- 8K RAM
 - 2 x 4K
- 2K EEPROM

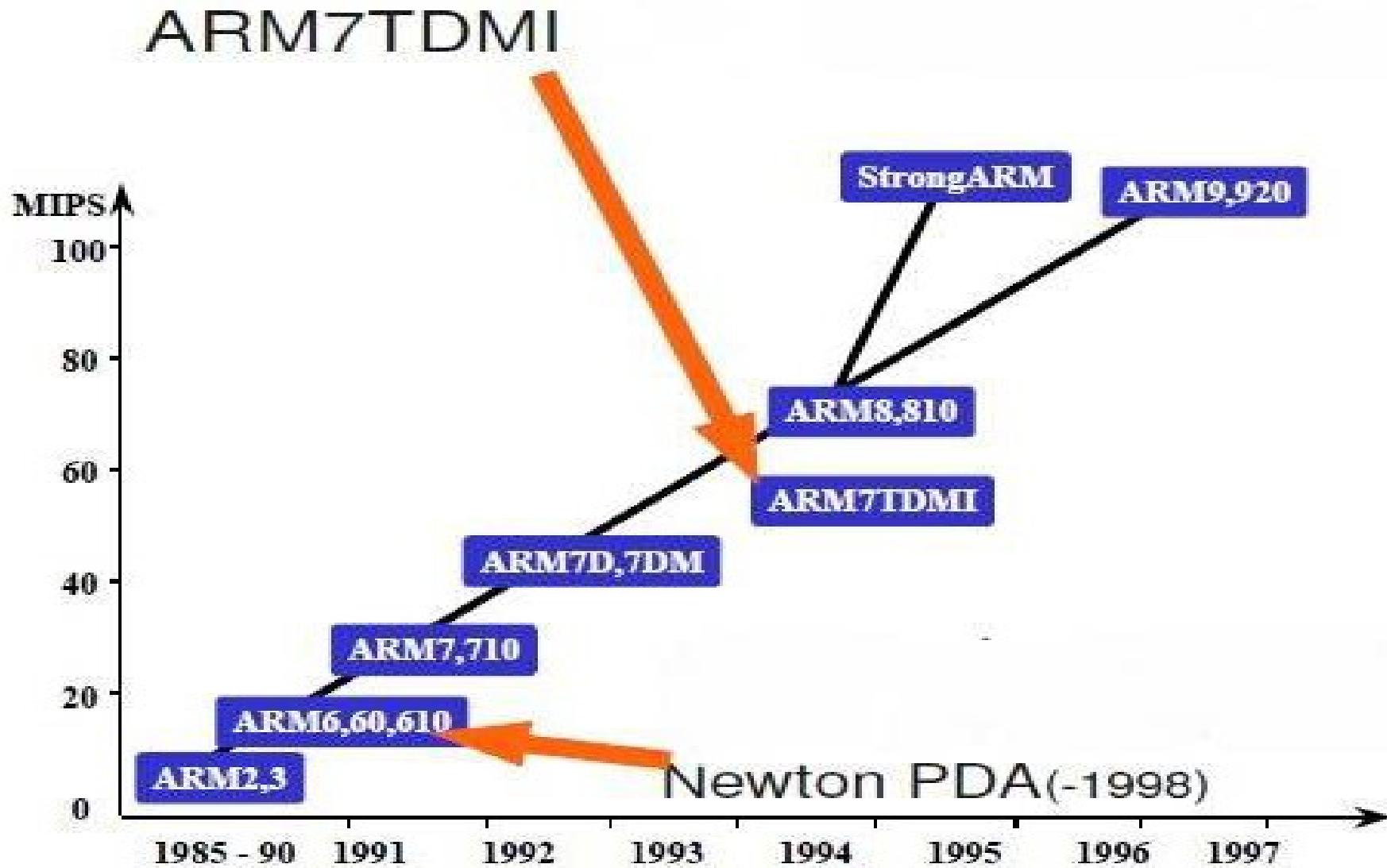


ARM Cortex Architecture

- Harvard architecture
 - Separate instructions and data



ARM History



Arm Processors

ARM Architecture: For Diverse Embedded Processing Needs

Cortex - A

Highest performance

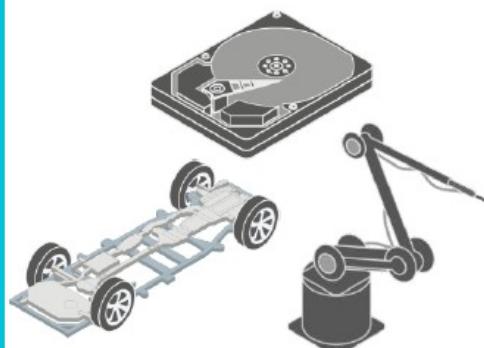
Optimised for
rich operating systems



Cortex - R

Fast response

Optimised for
high performance,
hard real-time applications



Cortex - M

Smallest/lowest power

Optimised for
discrete processing and
microcontrollers



Cortex family

Cortex-A8

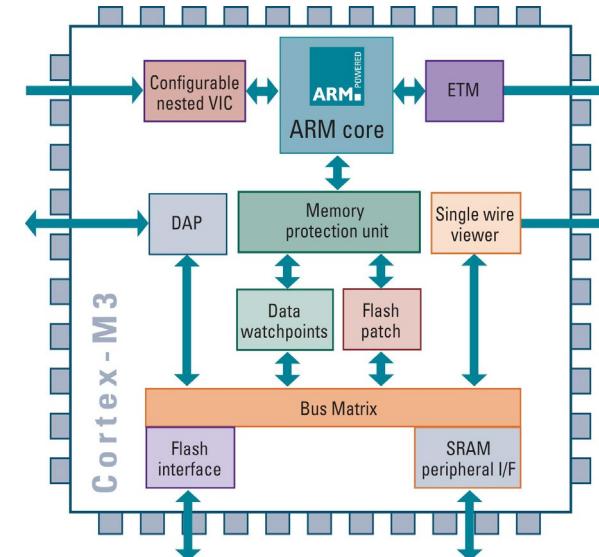
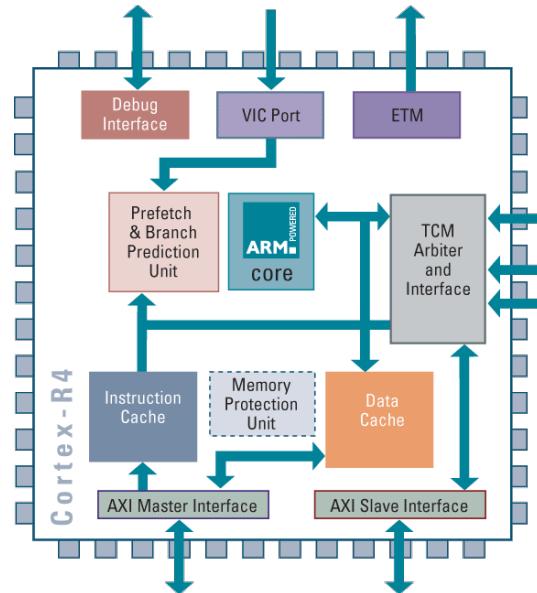
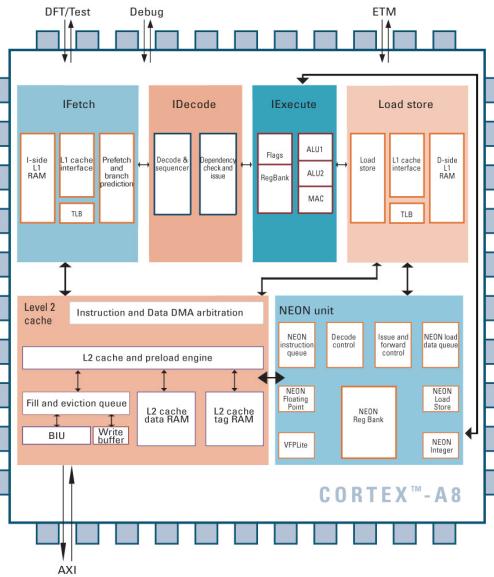
- Architecture v7A
 - MMU
 - AXI
 - VFP & NEON support

Cortex-R4

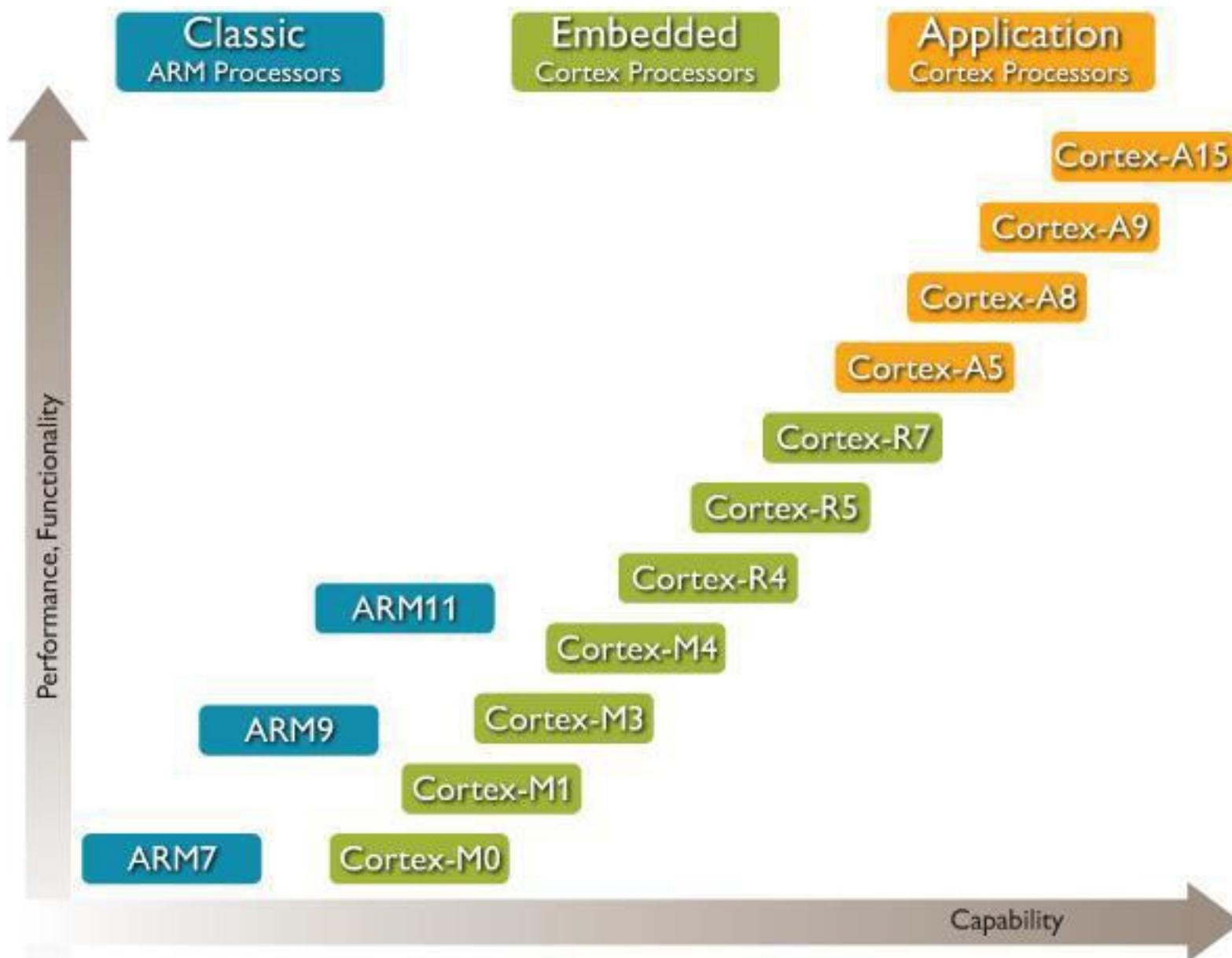
- Architecture v7R
 - MPU (optional)
 - AXI
 - Dual Issue

Cortex-M3

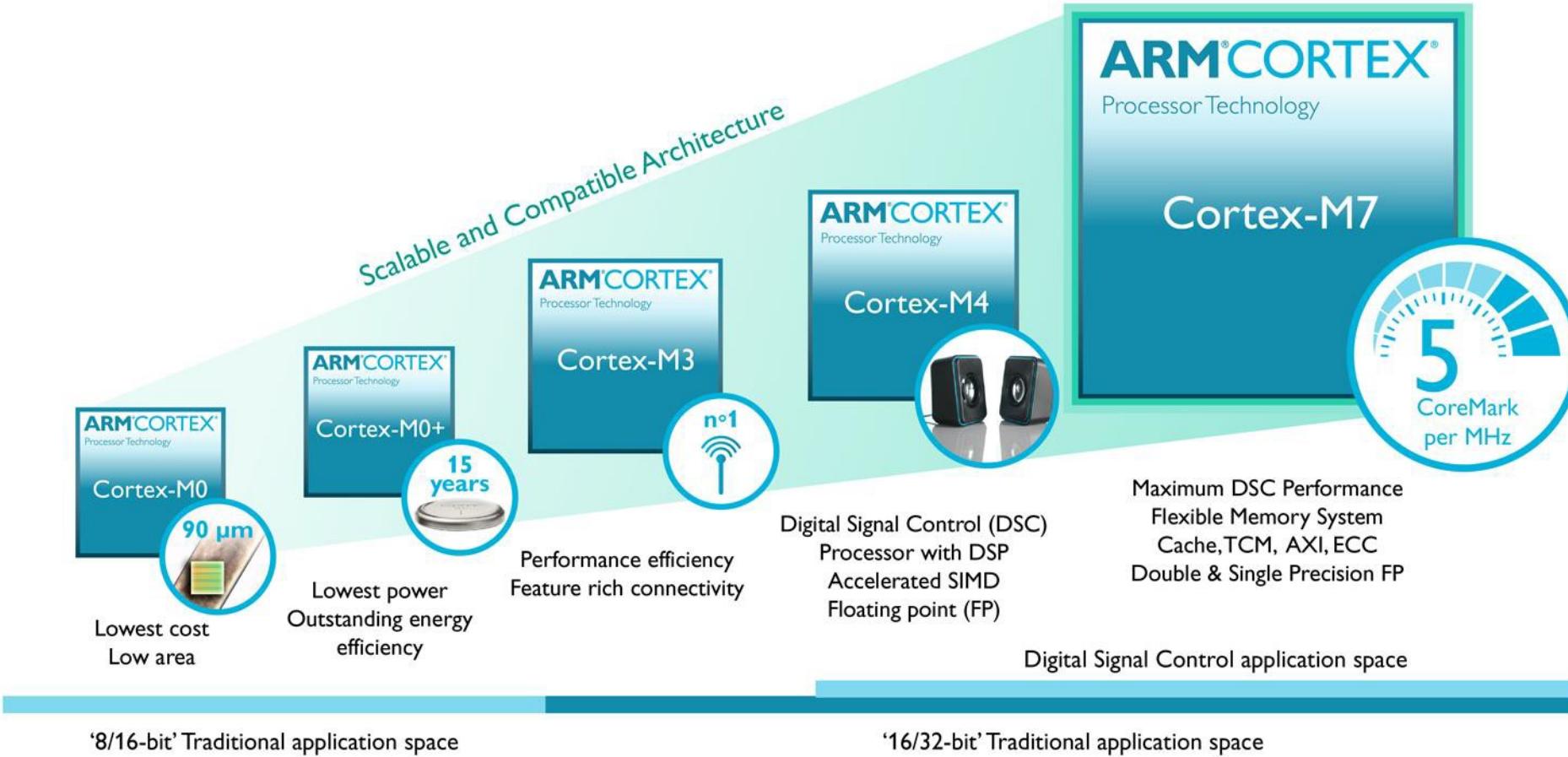
- Architecture v7M
 - MPU (optional)
 - AHB Lite & APB



Arm Processors

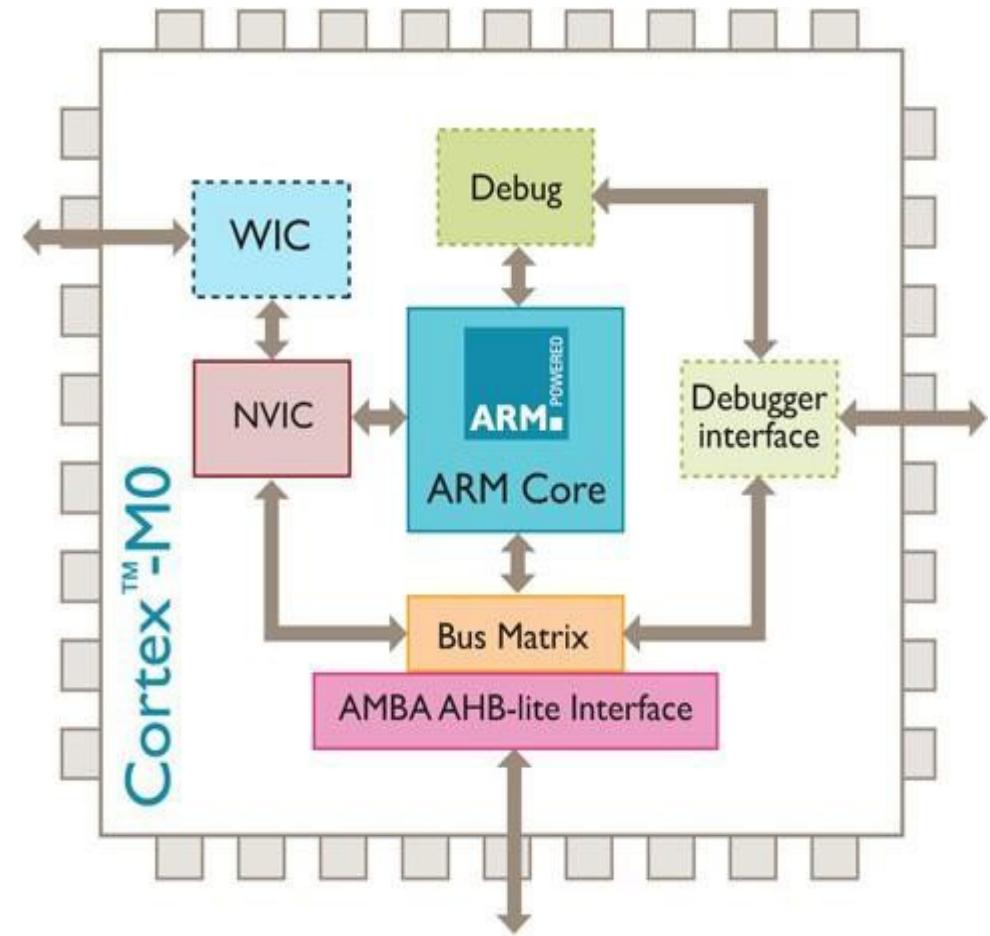


Cortex -M Processors



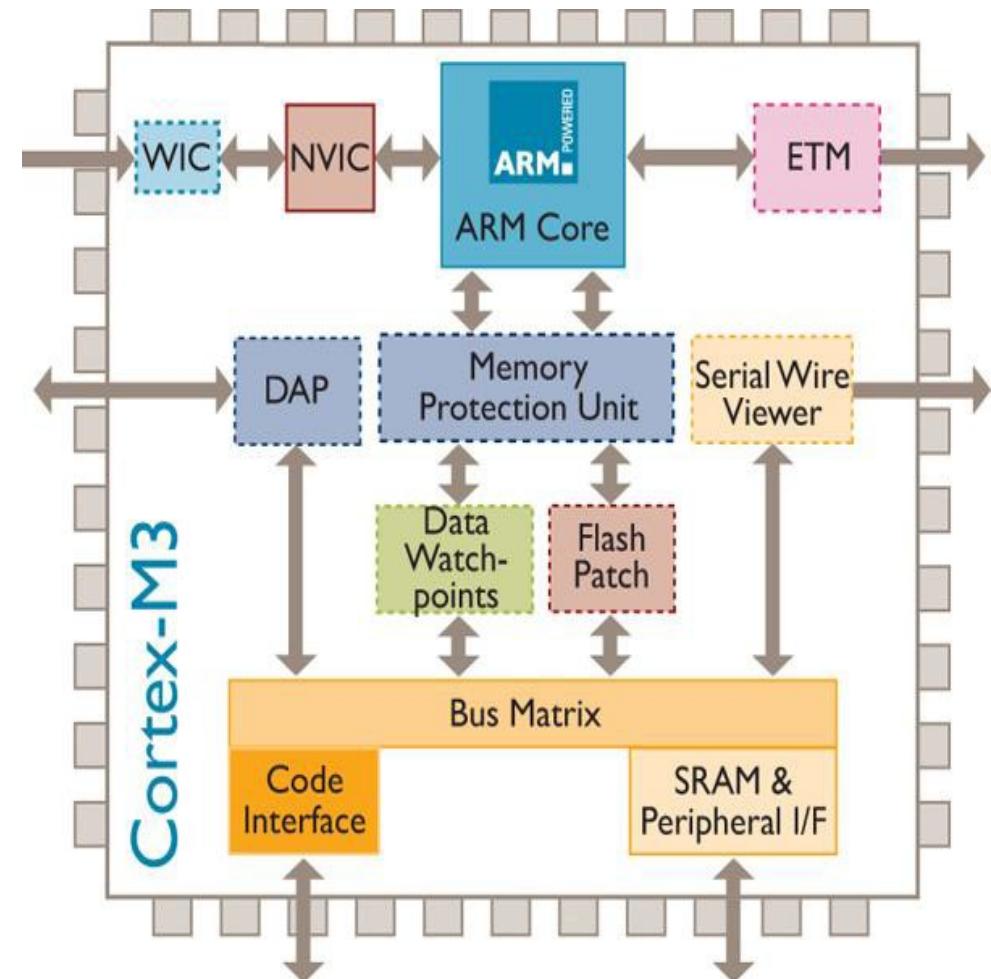
Embedded ARM Cortex Processors

- Cortex M0:
 - Ultra low gate count (less than 12K gates).
 - Ultra low-power ($3 \mu\text{W}/\text{MHz}$).
 - 32-bit processor.
 - Based on ARMv6-M architecture.



Embedded ARM Cortex Processors

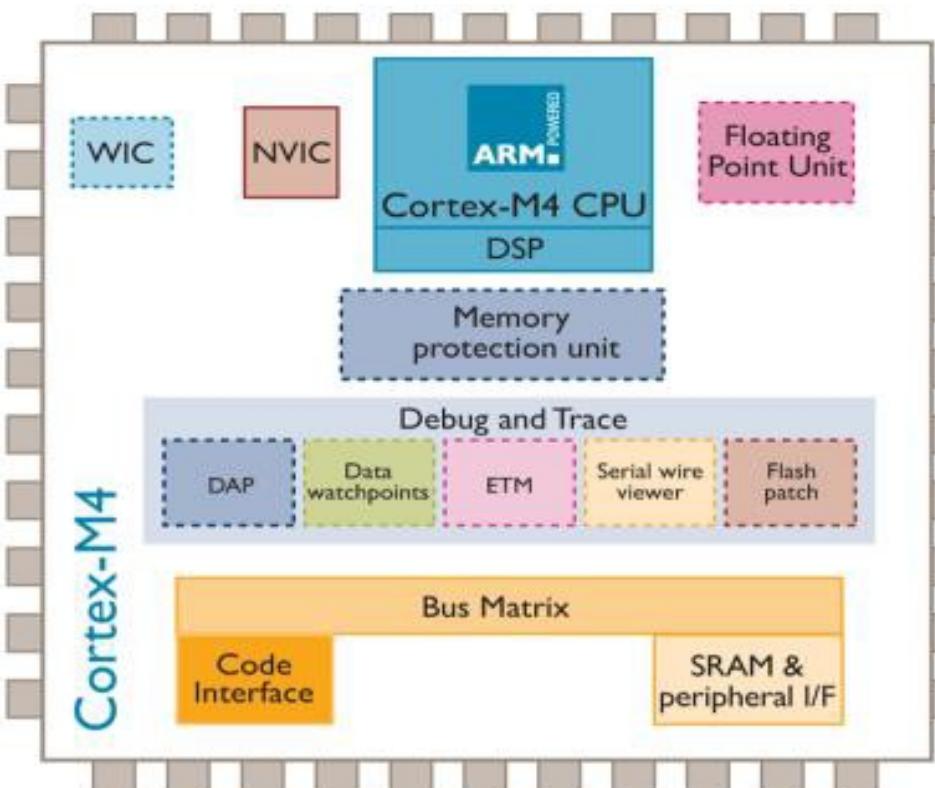
- Cortex M3:
 - The mainstream ARM processor for microcontroller applications.
 - High performance and energy efficiency.
 - Easy migration path from FPGA to ASIC.
 - Advanced 3-Stage Pipeline.
 - Based on ARMv7-M architecture.



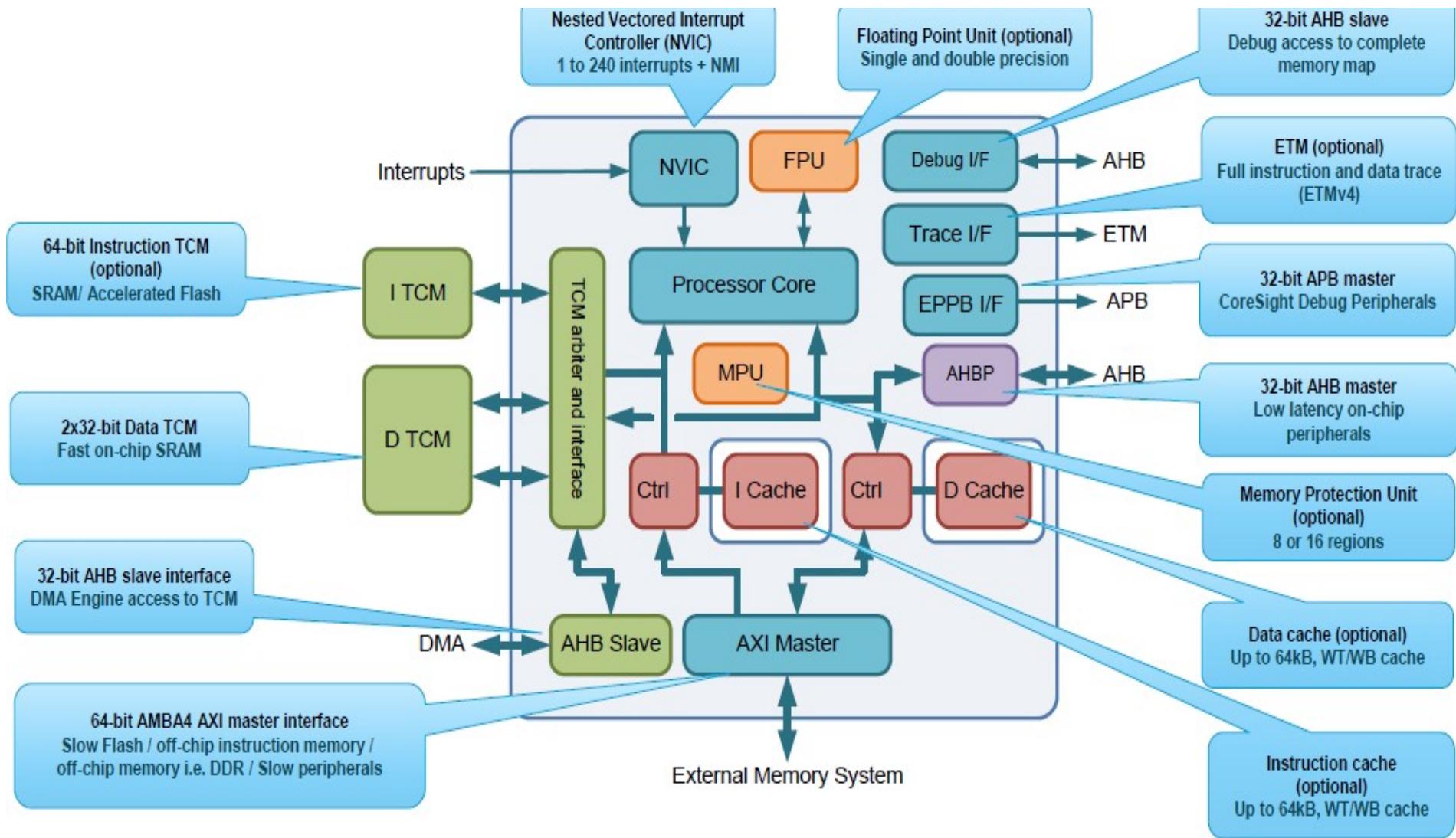
Embedded ARM Cortex Processors

- Cortex M4:

- Embedded processor for DSP.
- FPU (Floating Point Unit).
- Based on ARMv7E-M architecture.



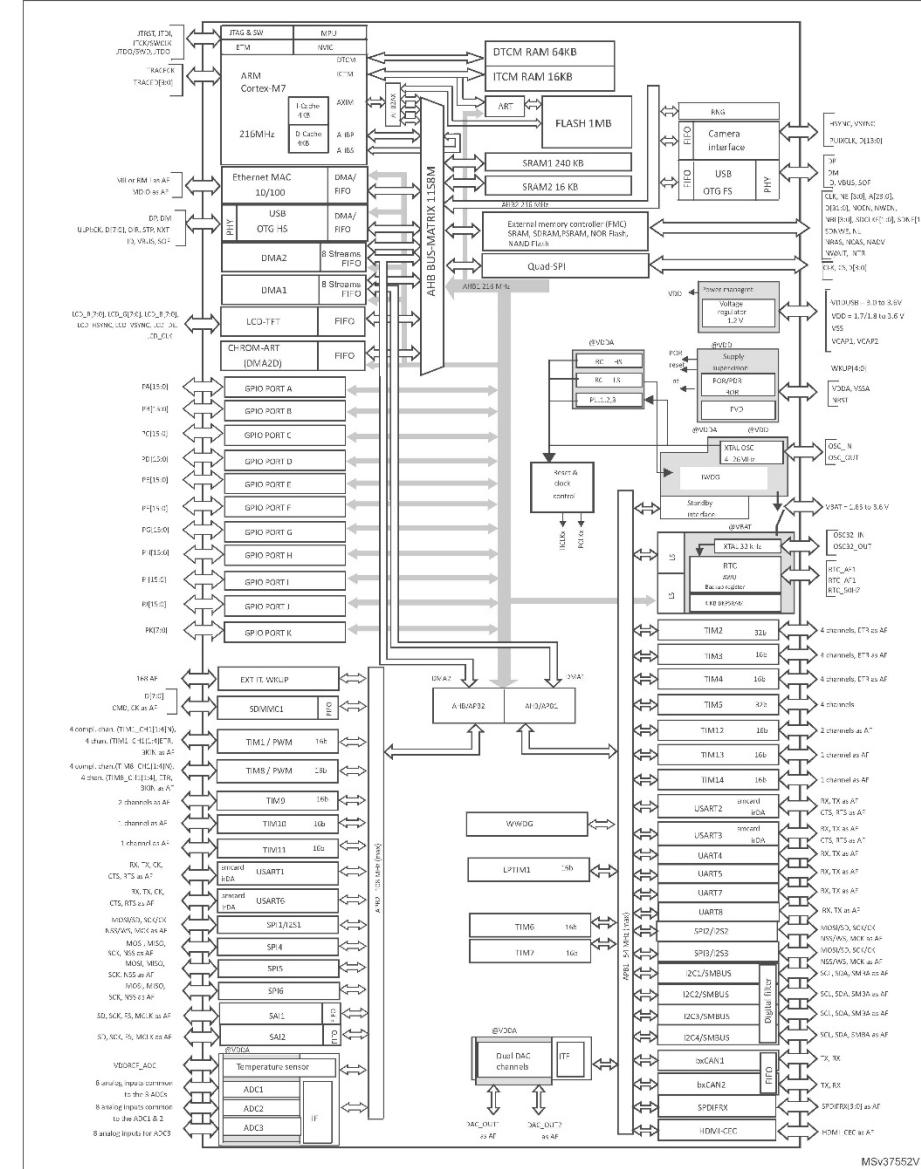
Cortex M7





STM32F746NG Block Diagram

Figure 2. STM32F745xx and STM32F746xx block diagram



Bus Matrix

Figure 3. STM32F745xx and STM32F746xx AXI-AHB bus matrix architecture

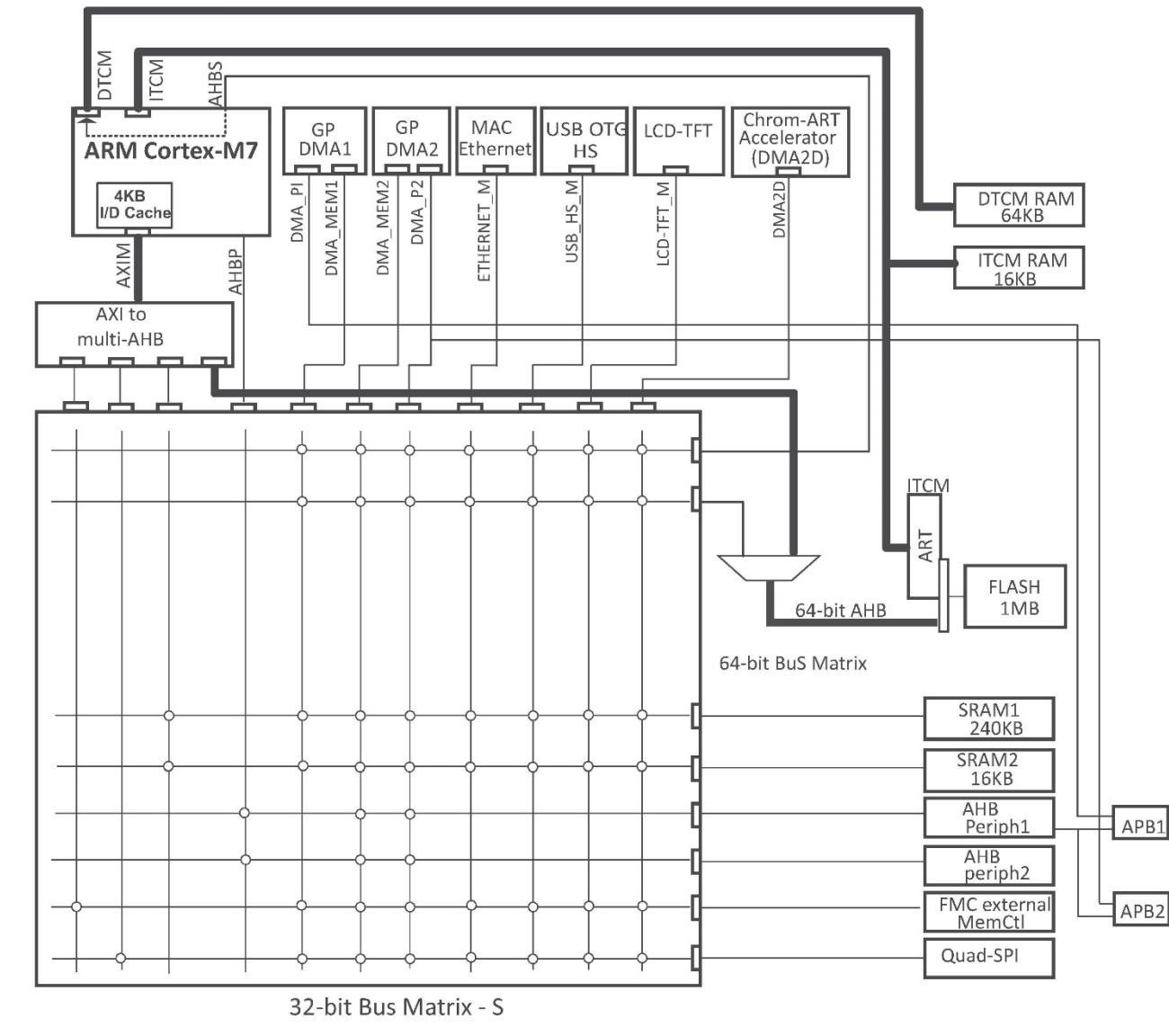
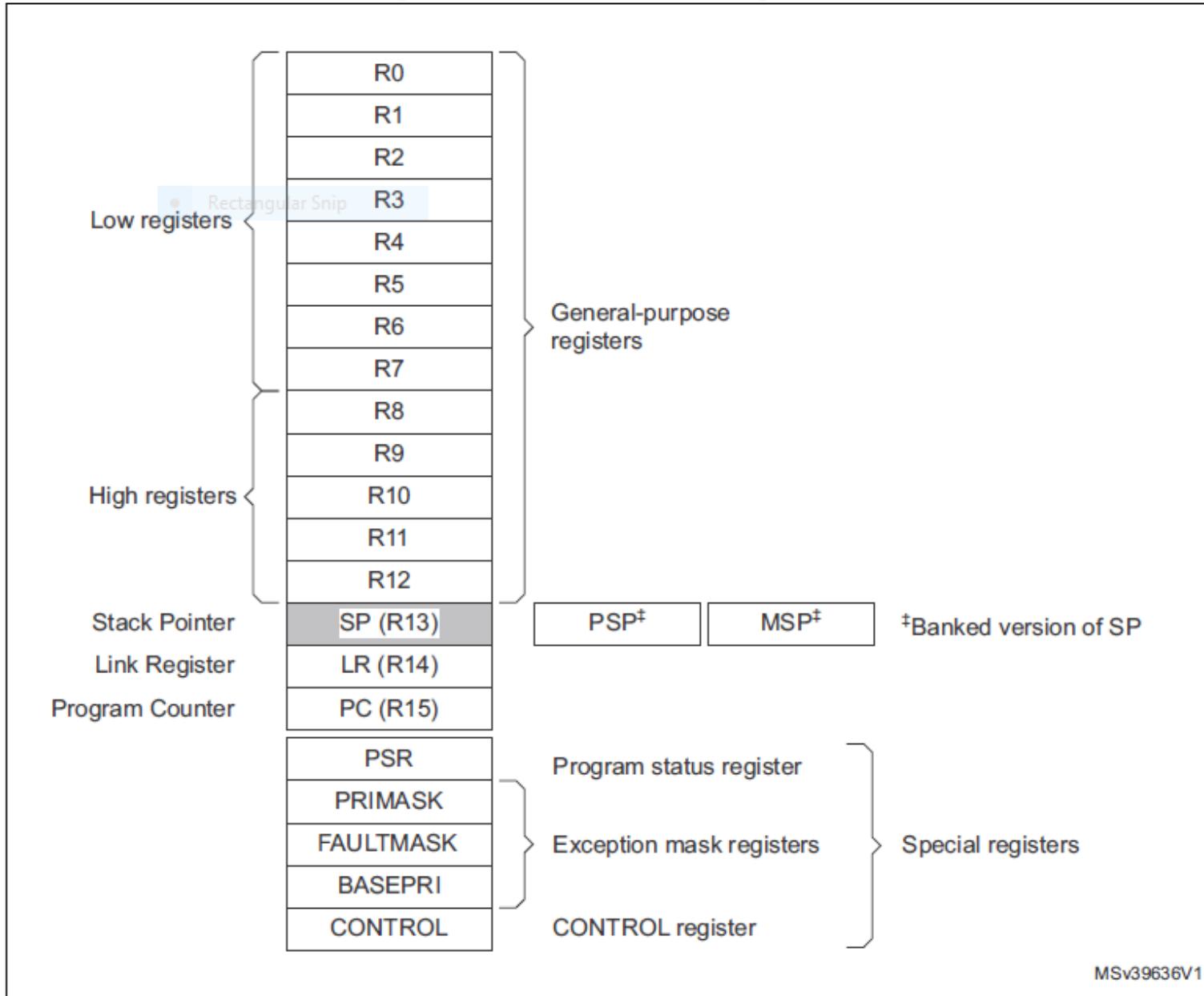


Figure 2. Processor core registers



Processor Core Registers

MSv39636V1

Program Status Register

Figure 3. APSR, IPSR and EPSR bit assignments

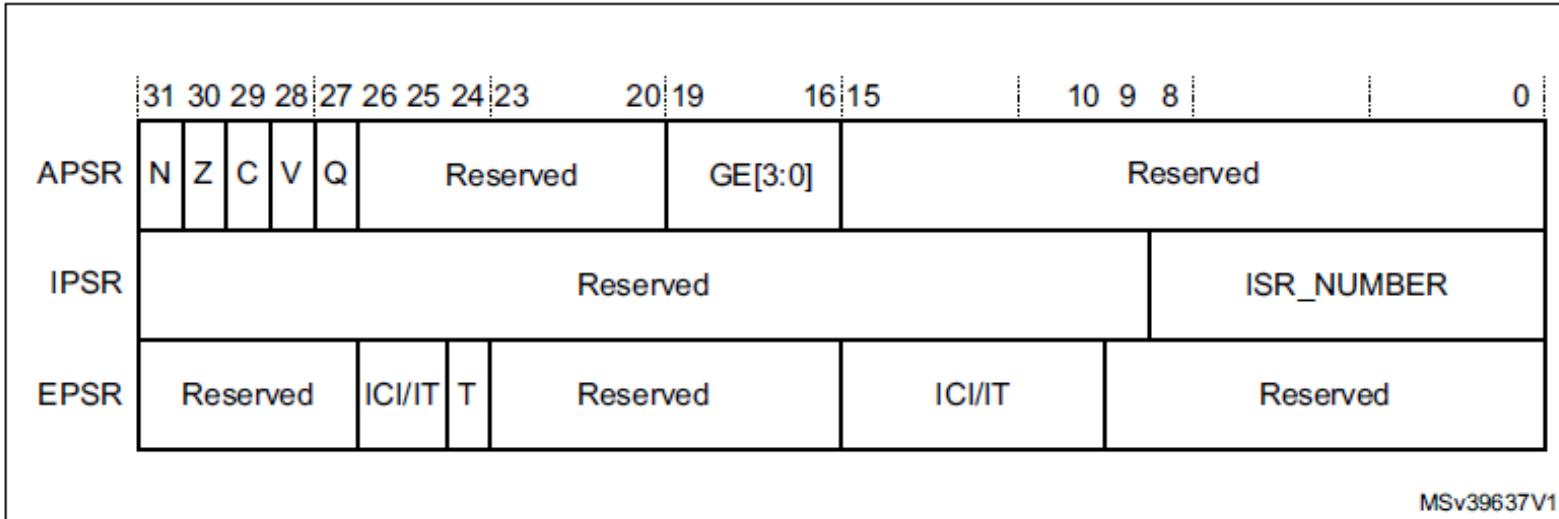


Table 4. APSR bit assignments

| Bits | Name | Description |
|------|------|-----------------------|
| [31] | N | Negative flag. |
| [30] | Z | Zero flag. |
| [29] | C | Carry or borrow flag. |
| [28] | V | Overflow flag. |

Memory

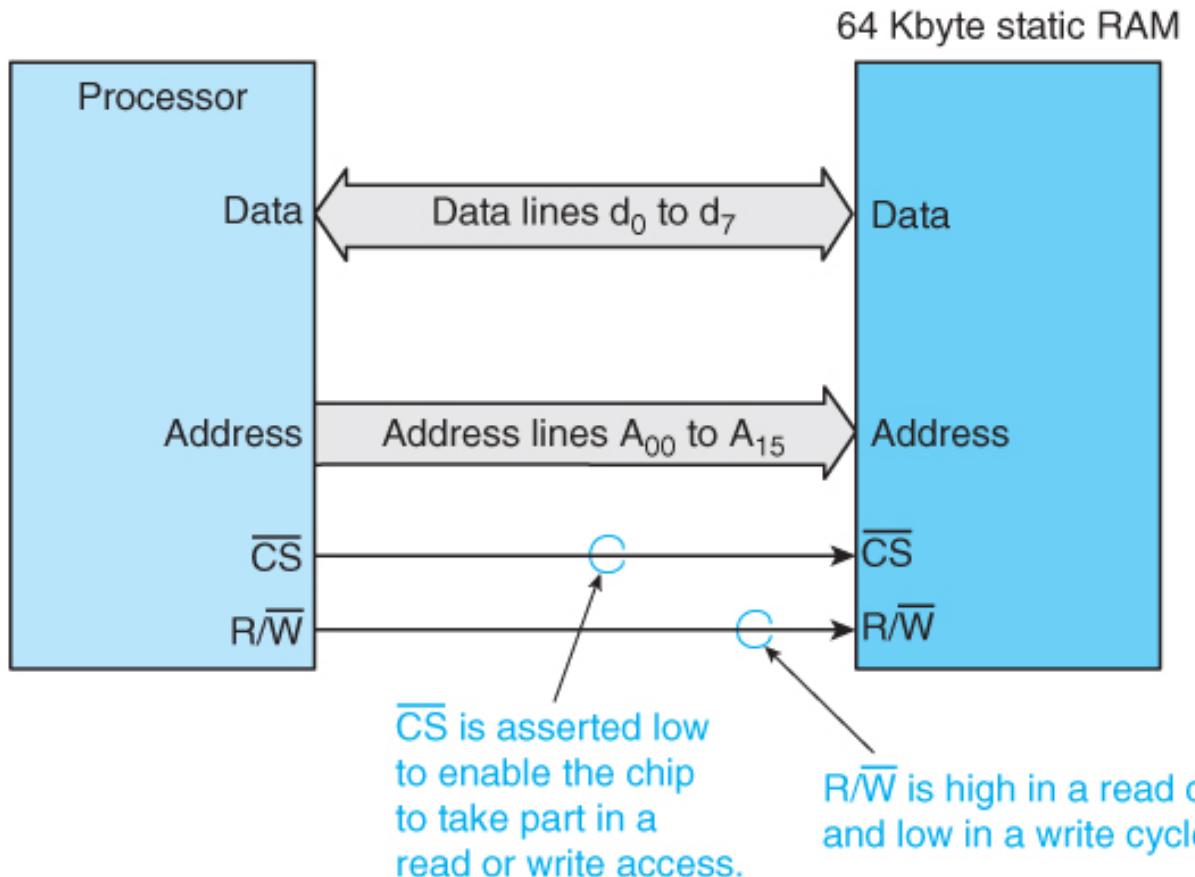
1. RAM (Random Access Memory)
 - Volatile
 - Static RAM (MOS/MOSFET)
 - Dynamic RAM (Capacitor)
 - Local Variable
2. ROM (Read Only Memory)
 - Nonvolatile
 - More complicated to write
 - Code + Constant

Memory vs Application

| Characteristic | DRAM | Static RAM | Flash memory |
|----------------|---------------|--------------|-------------------------|
| Static | No | Yes | Yes |
| Volatile | Yes | Yes | No |
| Typical size | 256 Mbits | 64 Mbits | 256 Mbits |
| Organization | 4 bits x 64 M | 8 bits x 8 M | 8 bits x 32 M |
| Access time | 10 ns | 2 ns | 40 ns |
| Application | Main store | Cache memory | BIOS, digital film, MP3 |

CPU & SRAM

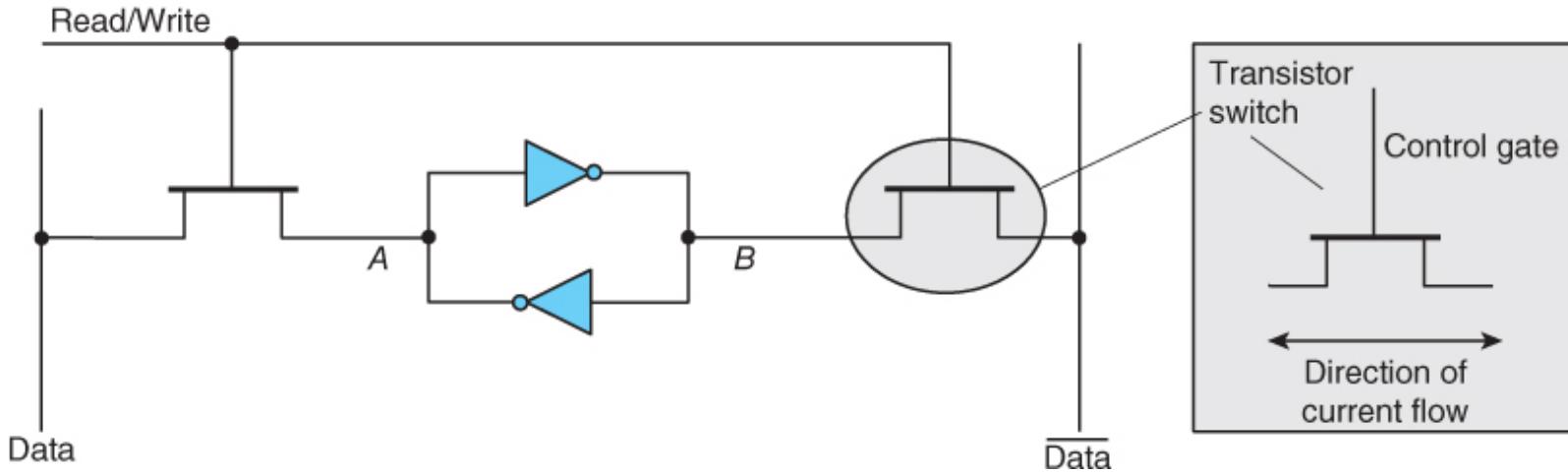
FIGURE 10.8 The static RAM chip



SRAM Cell

FIGURE 10.5

Operation of the static RAM cell



© Cengage Learning 2014

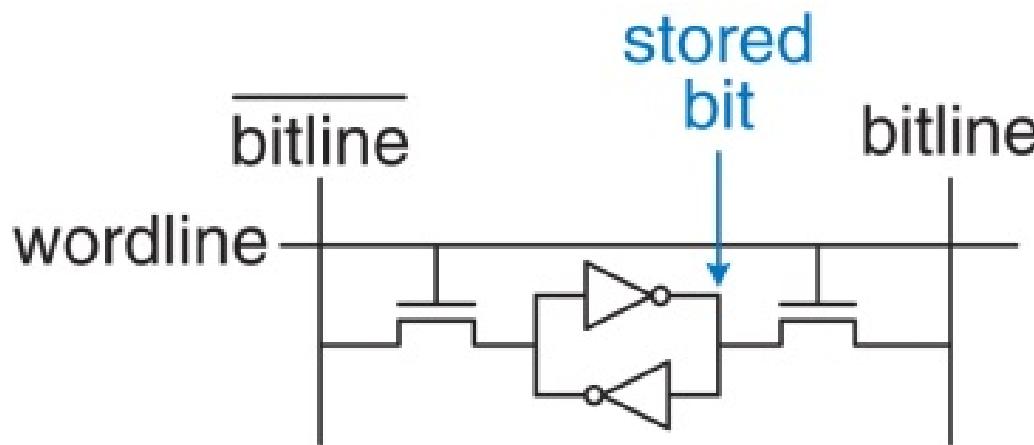
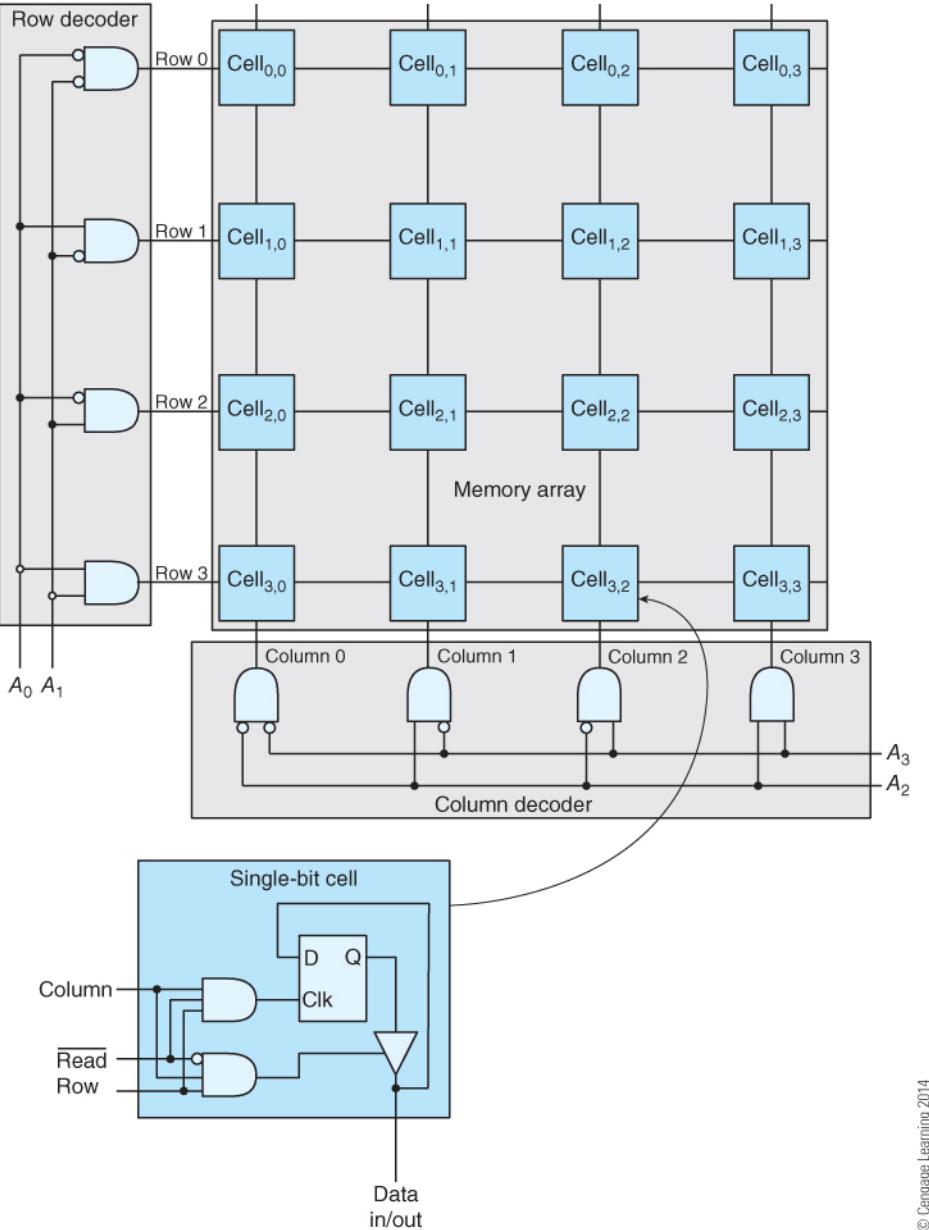


FIGURE 10.7 The arrangement of a static RAM array

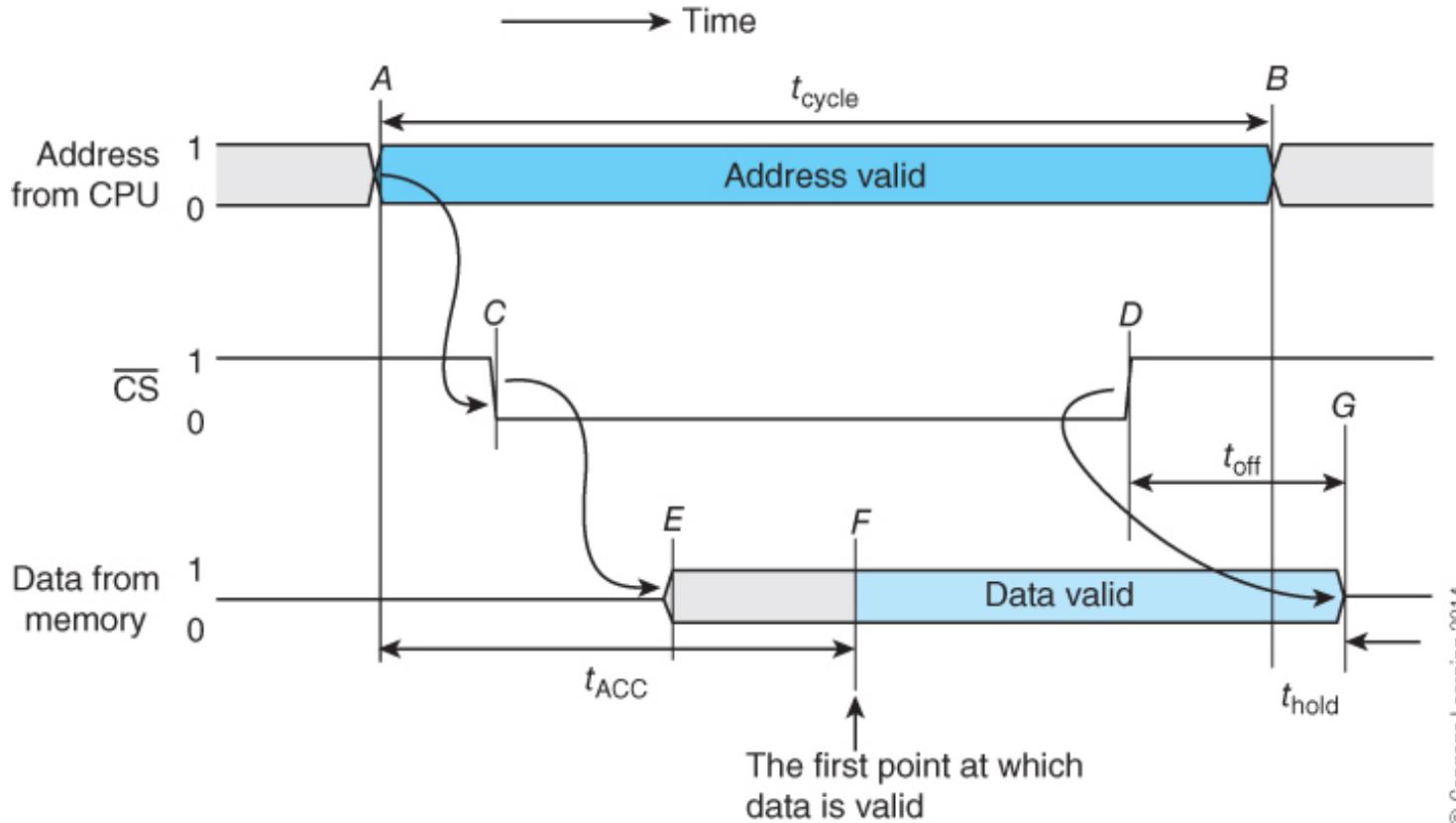


SRAM Cell & Structure

SRAM Read Cycle Timing

FIGURE 10.9

The static RAM read cycle timing diagram

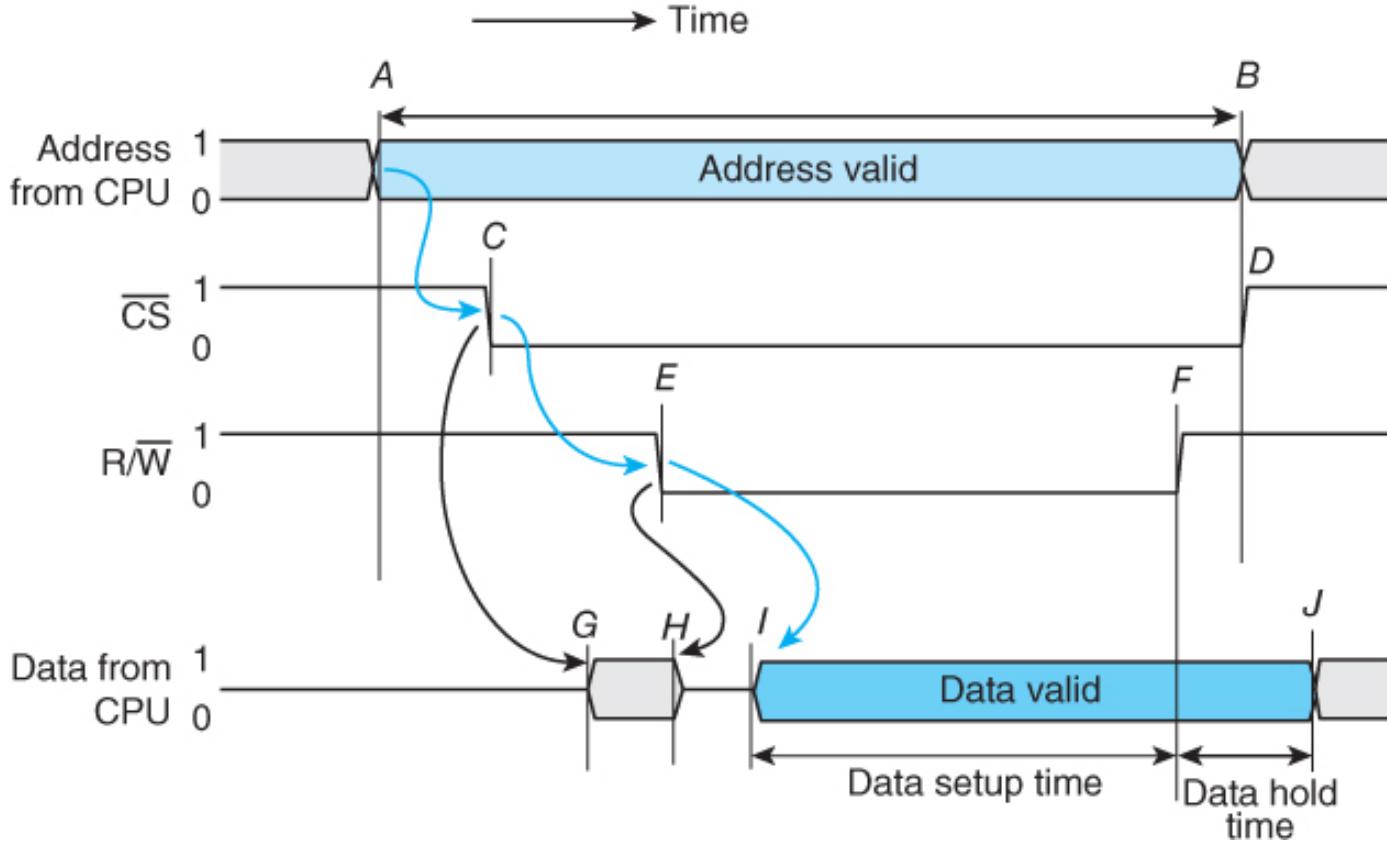


© Cengage Learning 2014

SRAM Write Cycle Timing

FIGURE 10.10

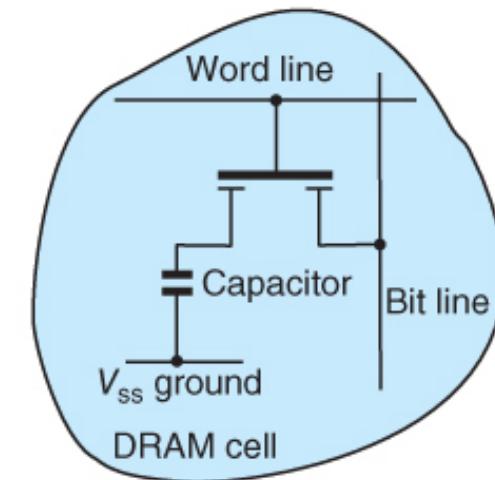
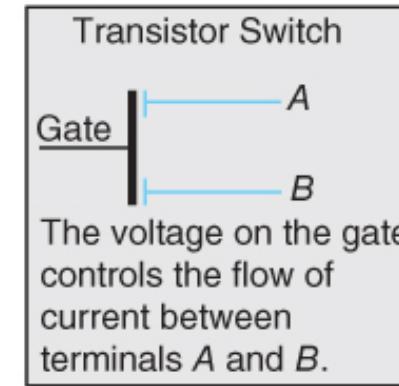
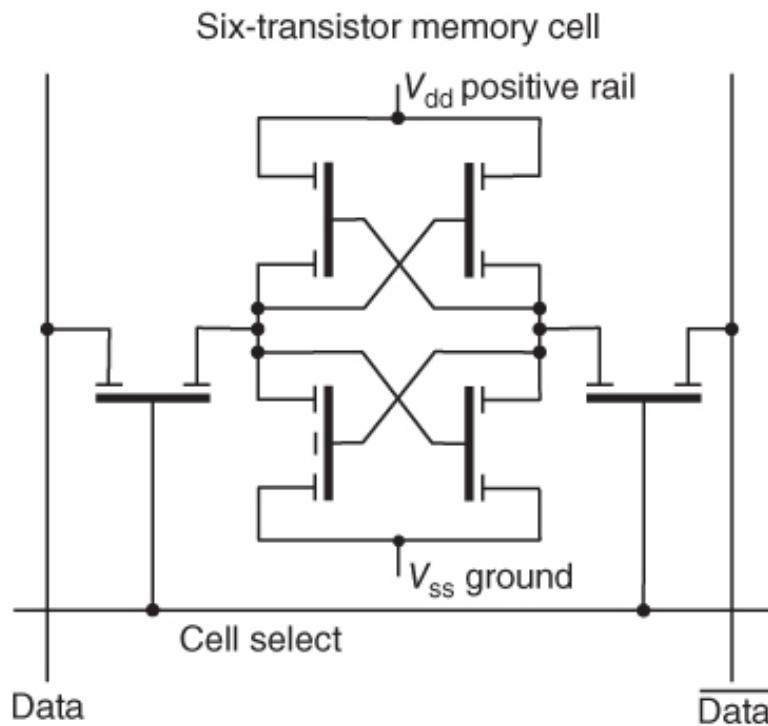
The static RAM write cycle timing diagram



SRAM Cell

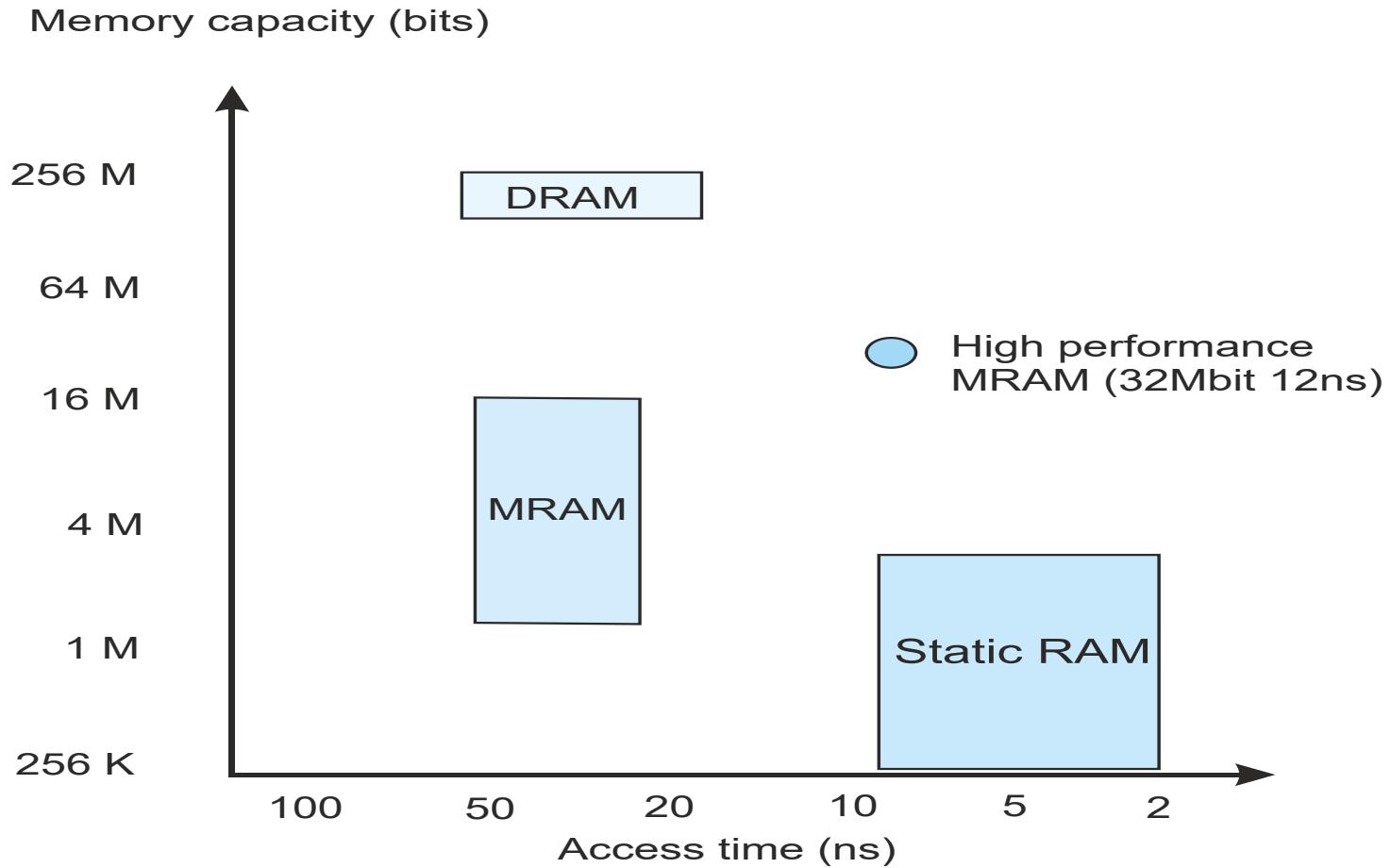
FIGURE 10.6

Circuit of the six-transistor static RAM cell with DRAM comparison

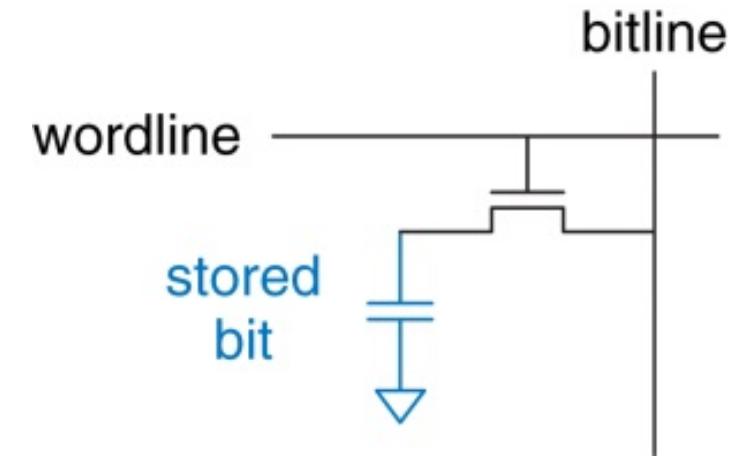
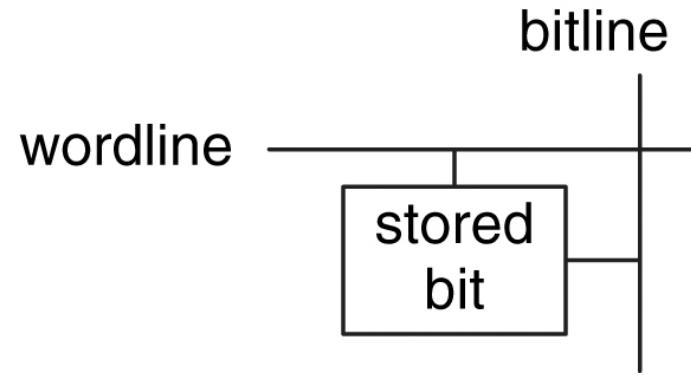


© Cengage Learning 2014

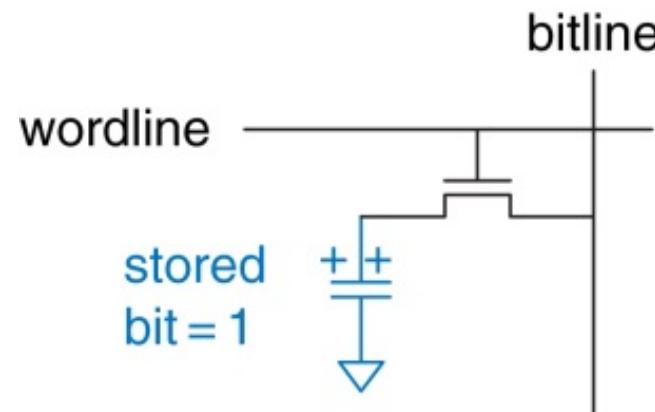
DRAM vs. SRAM Chip (Capacity & Time)



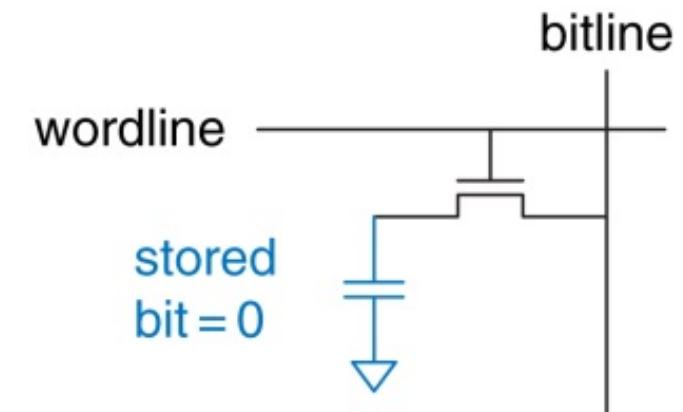
DRAM Bit Cell



DRAM Stored Values



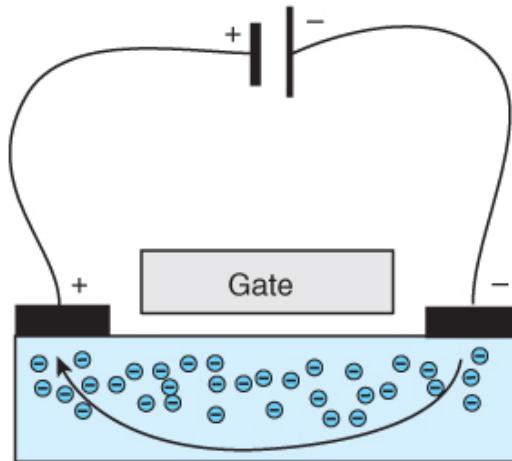
(a)



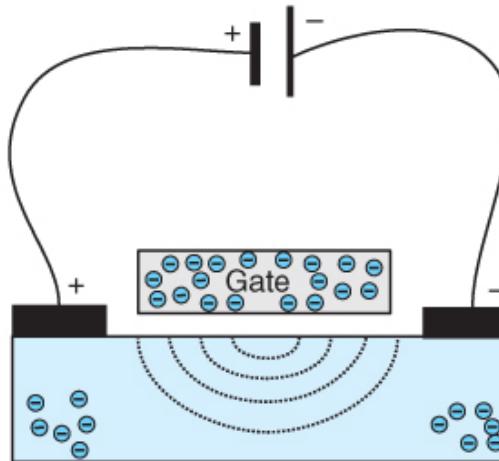
(b)

DRAM Cell

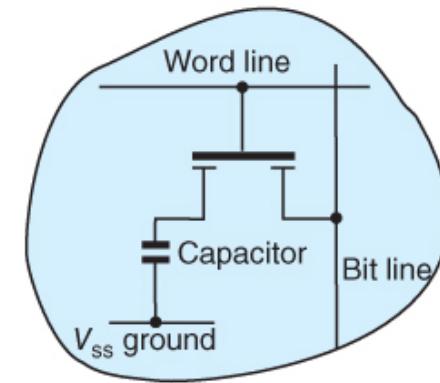
FIGURE 10.16 The effect of a charged gate on electron flow



(a) No charge on gate



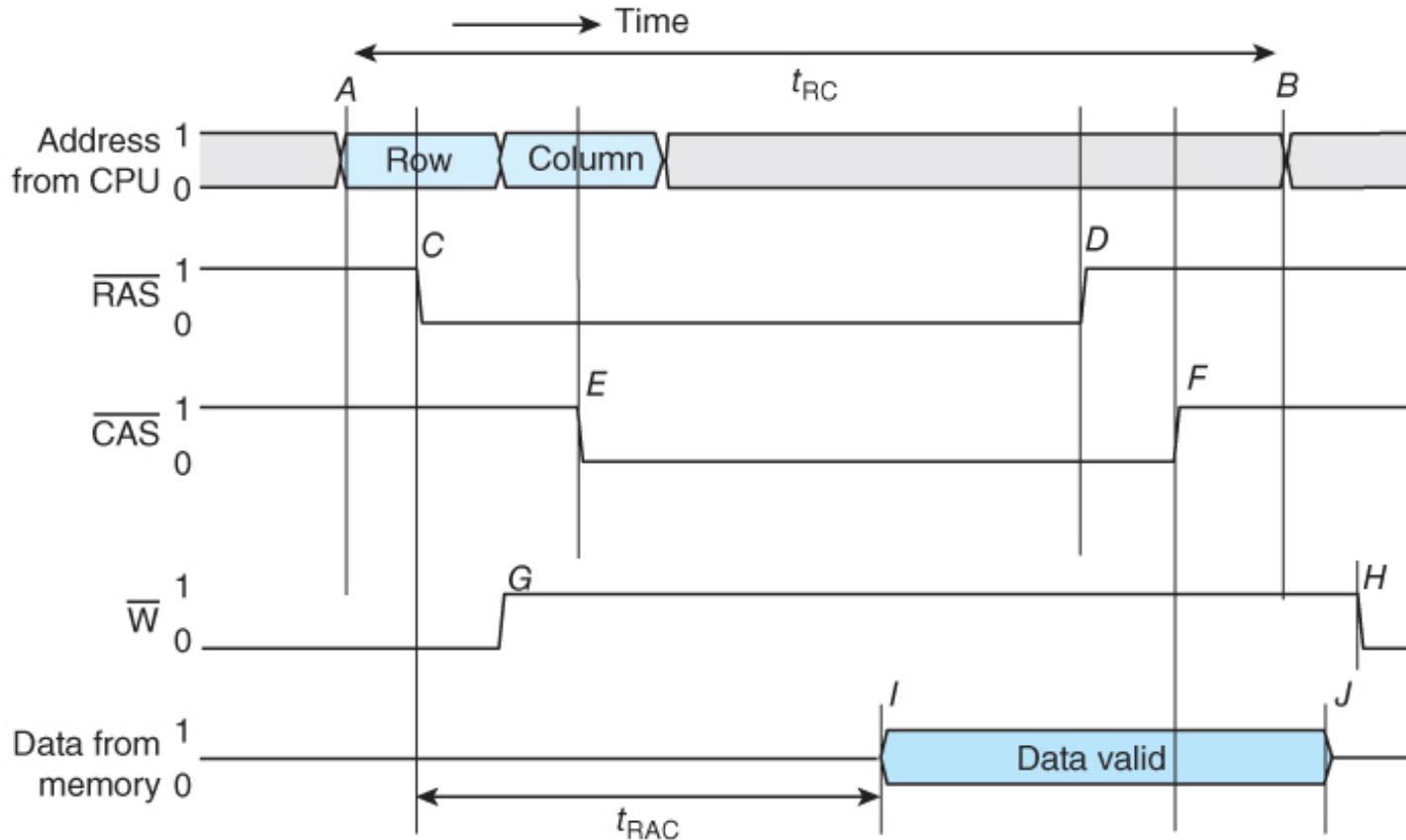
(b) Negative charge on gate



(c) DRAM cell

DRAM Read Cycle Timing

FIGURE 10.20 DRAM read cycle timing

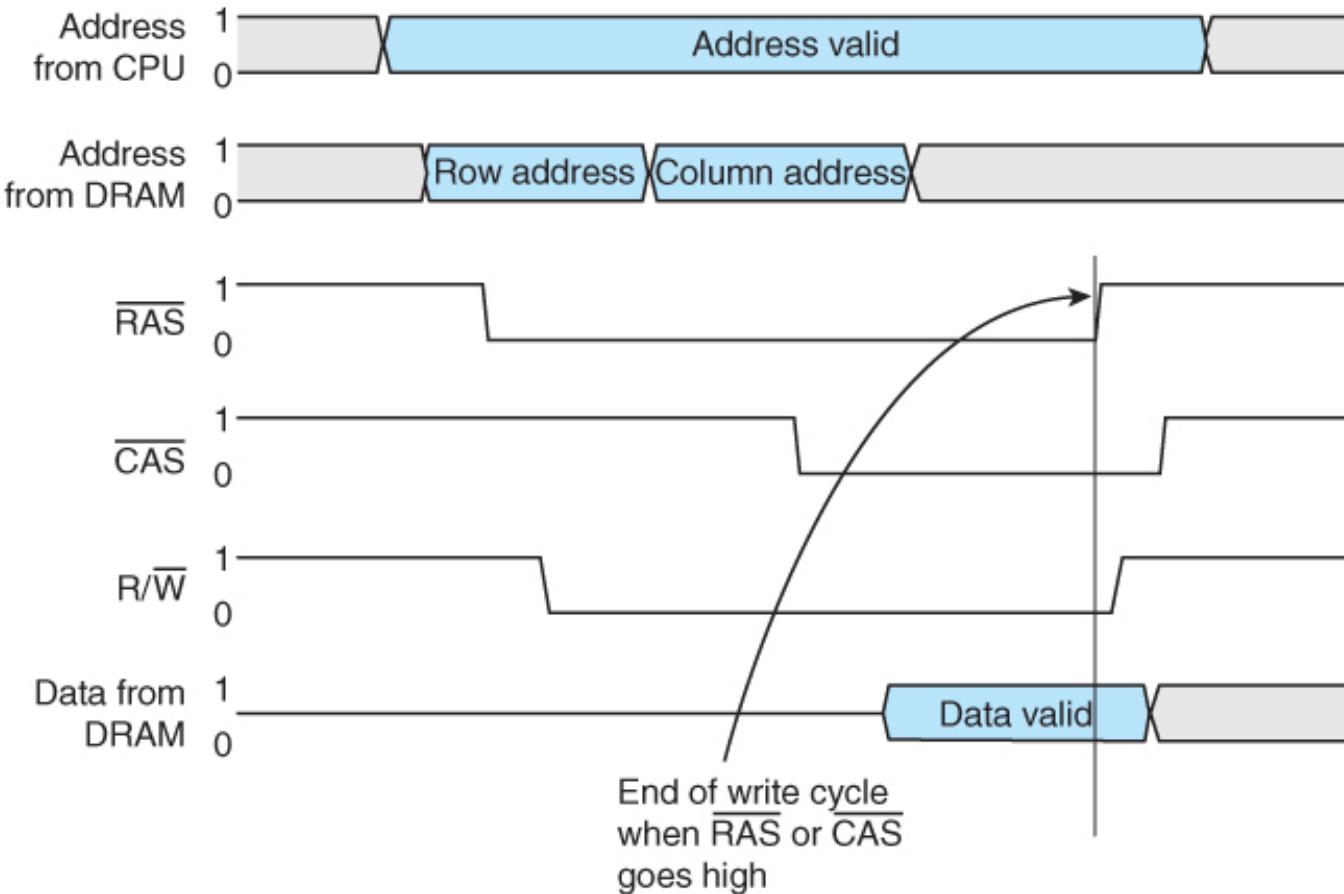


© Cengage Learning 2014

DRAM Write Cycle Timing

FIGURE 10.24

DRAM write cycle timing



DDR2-PC2-5300U 2GB 667-MHz DIMM (Dual Inline Memory Module)



Summary

- Simple register made from D-FF
 - Stores digital bits
- Cortex M is Harvard architecture
- Cortex M has 16 registers
 - 13 general purpose registers
 - 3 special registers SP, LR and PC
- N Z C V flag of Program Status Register
- Memory
 - RAM
 - ROM