Embedded Controller programming for Real Time Systems: ECE-40097

Lesson 2

Vijay Kumar

Main Topics

Processor Architecture

STM32L475V Processor

Processor Registers

Memory mapping

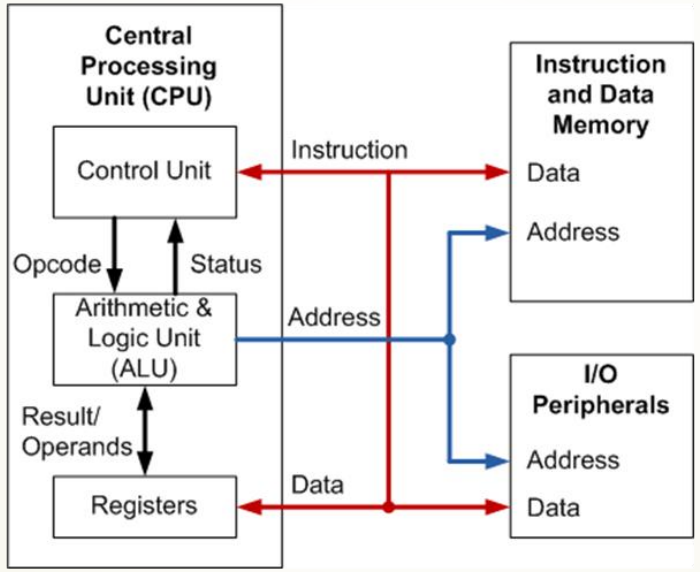
Code execution

Processor packaging

Processor Architecture

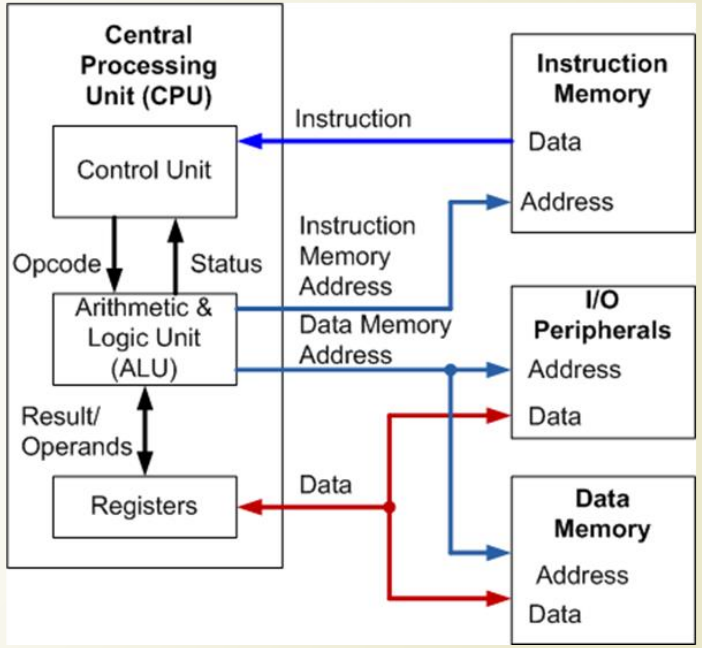
Von-Neumann

Instructions and data are stored in the same memory.

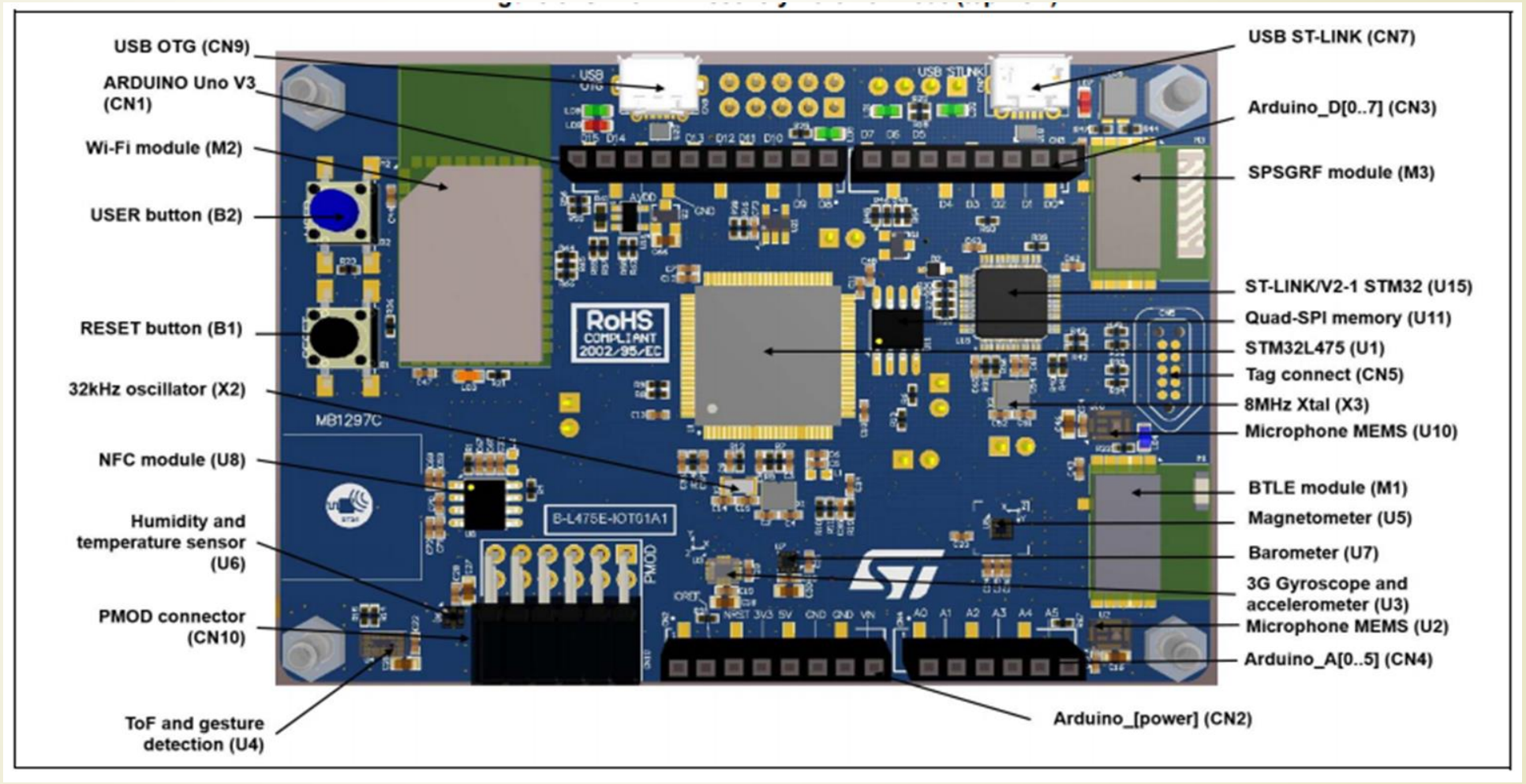


Harvard

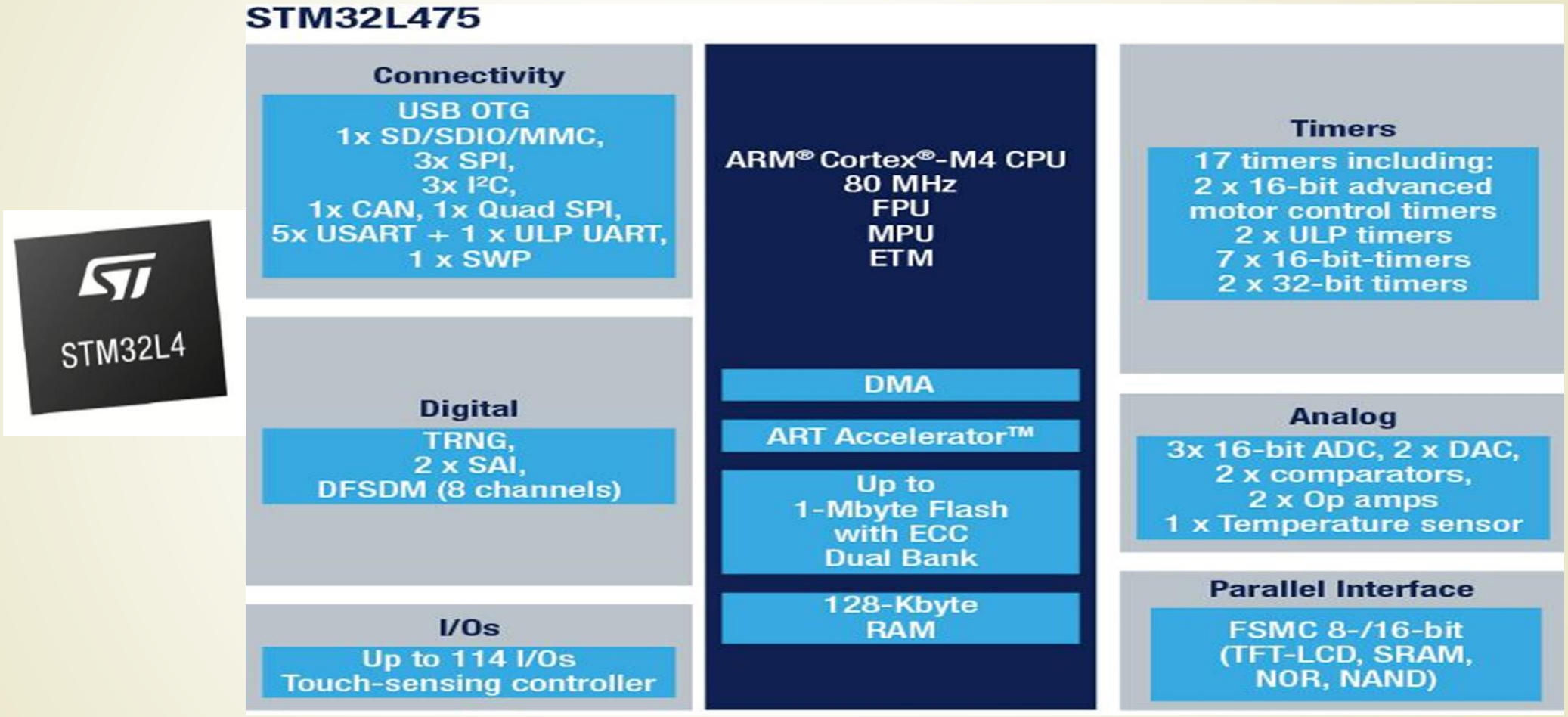
Data and instructions are stored into separate memories.



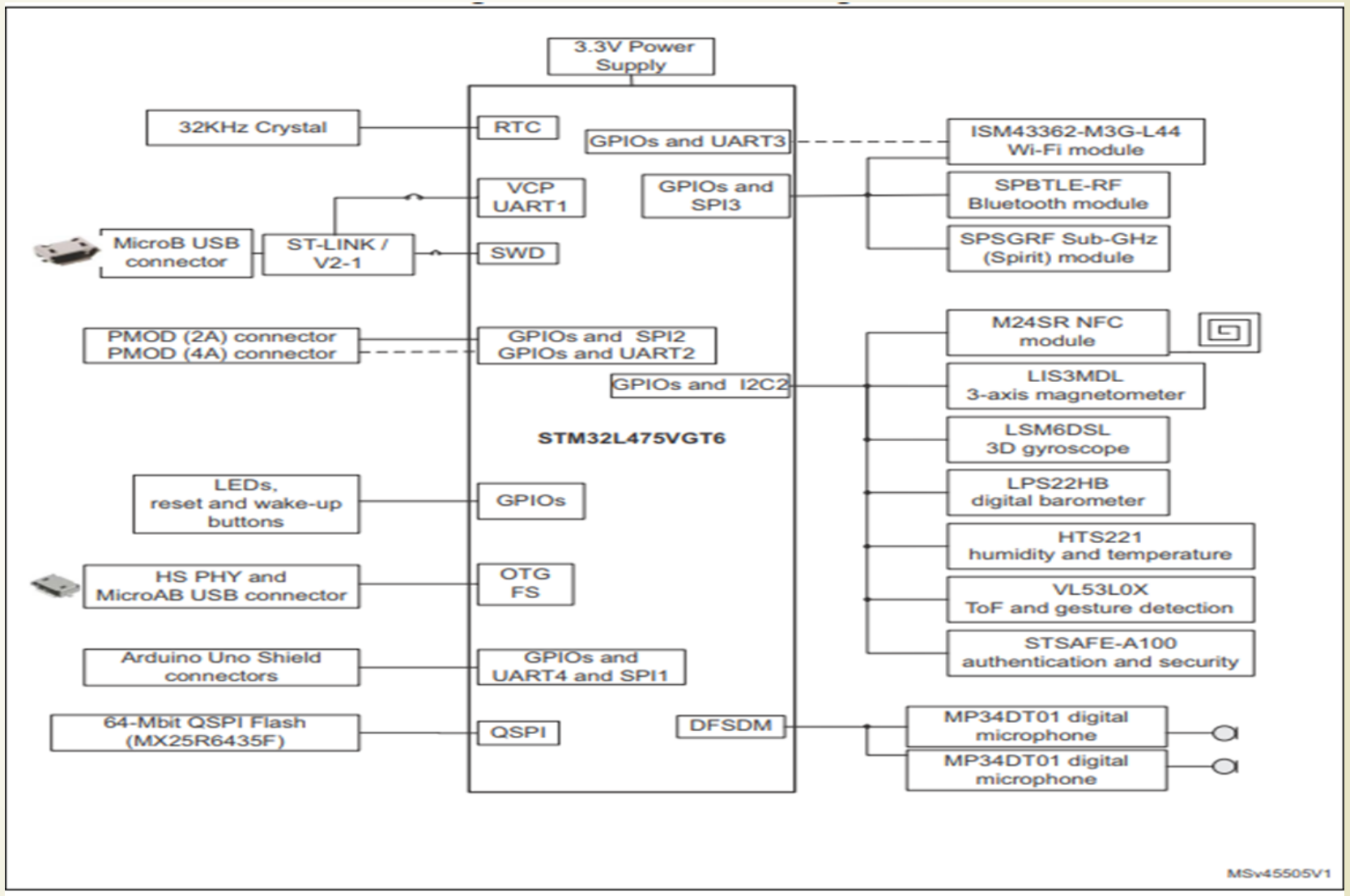
Development Board



Processor Peripherals



Block Diagram



About Processor

- Based of Cortex-M4 ARM Core
- Contains ARMv7E-M architecture
- Supports Harvard architecture
- Supports Thumb-1 and Thumb-2 instructions
  - More on this in next lesson
- Supports Saturated instructions
- 3 stages of instruction pipeline

About Processor

- Low latency interrupt processing (12 cycles)
- Nested vectored Interrupt Controller (NVIC)
- Max CPU Frequency 80 MHz
- Multiple timers
  - Including watchdog timers (IWDG and WWDG)
- Ultra low power with Flex power control
- Development support
  - Serial wire debug (SWD), JTAG

Connectivity

USART

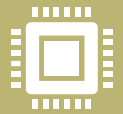
High speed high power

UART/LPUART

Low speed
Low Power

Controller are network (CAN)

Used primarily in vehicles



SPI/I2C

Serial programmable device (Higher data rate)
Inter integrated circuit interface

MPU

- Used for memory protection
  - To prevent accidental corruption of memory by any task
- Prevents a process from accessing memory that has not been allocated to
- Allows the privileged software to define memory regions
  - E.g. Kernel, Drivers
- Protection size could vary from 32 bytes to 4 Gb
- Is optional and could be bypassed if not needed
- But needed for system performance and data protection

Memory

Up to 1 Mbyte of Flash

For storing programs and data

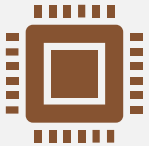
For dual bank boot

Up to 128 Kbyte of embedded SRAM

Boot Modes

From user Flash

E.g. USB or other types



From System memory

Place for boot loader

Programmed via UART, I2C, JTAG

From embedded SRAM

Power Modes

- 7 low power modes
- Wakeup based in interrupt
- Peripherals power management
- Usage depends on application types
- Highly desirable features for embedded system

Clocks and Startup

Prescalar to trade off between speed and power consumption

Option to change the clock frequency on the fly

Depends on the application

Recommend to stay at desired clock and minimize the switch

Types of I/O

Memory-mapped I/O (MMIO)

Uses memory to map Memory and IO devices

Port-mapped I/O (PMIO)

Uses special instructions for doing I/O

Examples are in, out

Memory Map

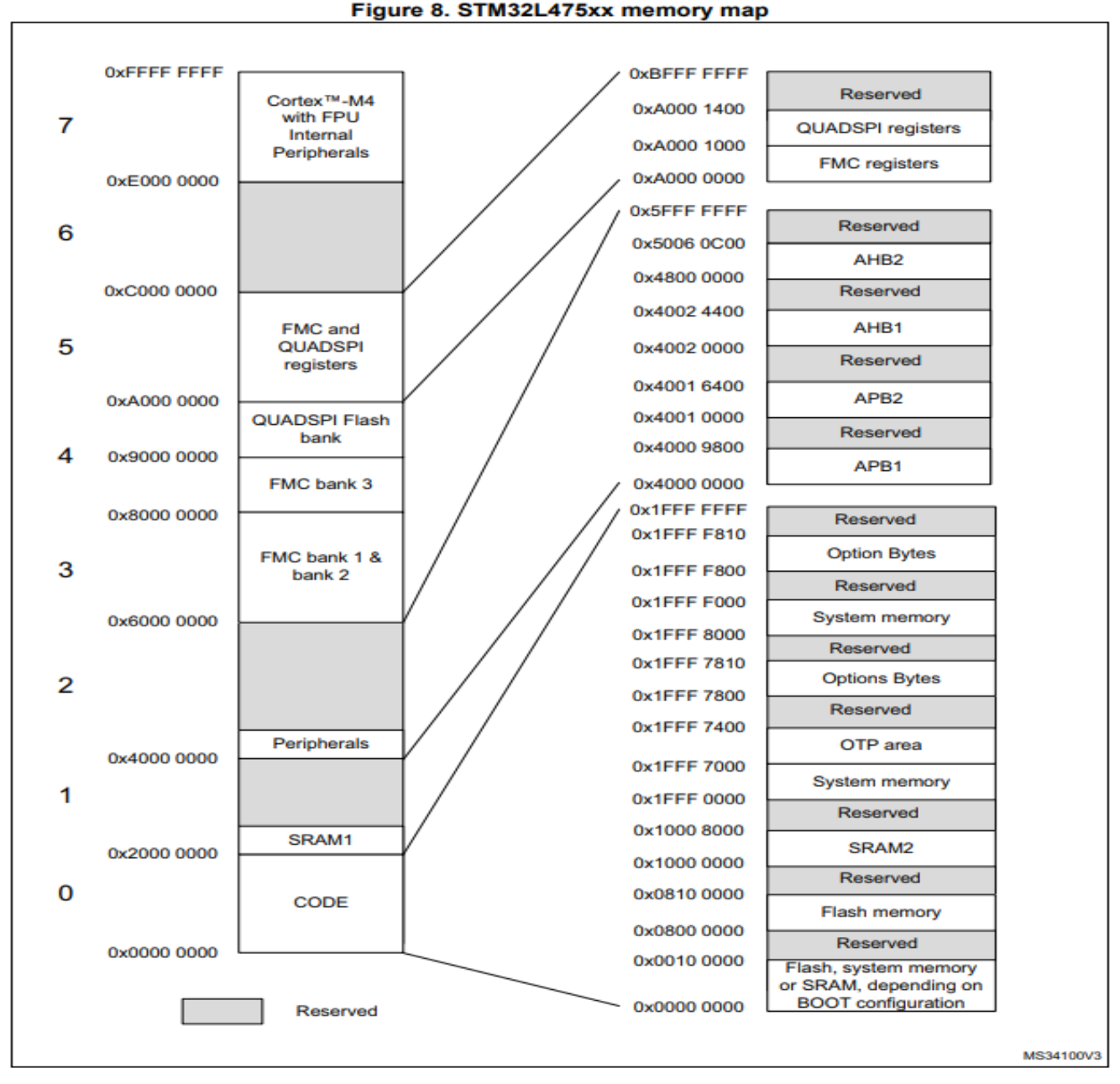


Figure 8. STM32L475xx memory map
0xFFFF FFFF
7
0xE000 0000
6
0xC000 0000
5
0xA000 0000
4
0x9000 0000
0x8000 0000
3
0x6000 0000
2
0x4000 0000
1
0x2000 0000
0
0x0000 0000
Cortex™-M4 with FPU Internal Peripherals
FMC and QUADSPI registers
QUADSPI Flash bank
FMC bank 3
FMC bank 1 & bank 2
Peripherals
SRAM1
CODE
Reserved
0xBFFF FFFF
0xA000 1400
0xA000 1000
0xA000 0000
0x5FFF FFFF
0x5006 0C00
0x4800 0000
0x4002 4400
0x4002 0000
0x4001 6400
0x4001 0000
0x4000 9800
0x4000 0000
0x1FFF FFFF
0x1FFF F810
0x1FFF F800
0x1FFF F000
0x1FFF 8000
0x1FFF 7810
0x1FFF 7800
0x1FFF 7400
0x1FFF 7000
0x1FFF 0000
0x1000 8000
0x1000 0000
0x0810 0000
0x0800 0000
0x0010 0000
0x0000 0000
Reserved
QUADSPI registers
FMC registers
Reserved
AHB2
Reserved
AHB1
Reserved
APB2
Reserved
APB1
Reserved
Option Bytes
Reserved
System memory
Reserved
Options Bytes
Reserved
OTP area
System memory
Reserved
SRAM2
Reserved
Flash memory
Reserved
Flash, system memory or SRAM, depending on BOOT configuration
MS34100V3

Base Address Memory Map

- #define FLASH_BASE (0x08000000UL) /*!< FLASH(up to 1 MB) base address */
- #define SRAM1_BASE (0x20000000UL) /*!< SRAM1(up to 96 KB) base address */
- #define SRAM2_BASE (0x10000000UL) /*!< SRAM2(32 KB) base address */
- #define PERIPH_BASE (0x40000000UL) /*!< Peripheral base address */
- #define FMC_BASE (0x60000000UL) /*!< FMC base address */
- #define QSPI_BASE (0x90000000UL) /*!< QUADSPI memories accessible over AHB base address */

- #define FMC_R_BASE (0xA0000000UL) /*!< FMC control registers base address */
- #define QSPI_R_BASE (0xA0001000UL) /*!< QUADSPI control registers base address */
- #define SRAM1_BB_BASE (0x22000000UL) /*!< SRAM1(96 KB) base address in the bit-band region */
- #define PERIPH_BB_BASE (0x42000000UL) /*!< Peripheral base address in the bit-band region */

Peripheral Memory Map

- #define APB1PERIPH_BASE PERIPH_BASE
- #define APB2PERIPH_BASE (PERIPH_BASE + 0x00010000UL)
- #define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000UL)
- #define AHB2PERIPH_BASE (PERIPH_BASE + 0x08000000UL)

Memory Map Example

- #define TIM2_BASE (APB1PERIPH_BASE + 0x0000UL)
- #define TIM3_BASE (APB1PERIPH_BASE + 0x0400UL)
- #define TIM4_BASE (APB1PERIPH_BASE + 0x0800UL)
- #define TIM5_BASE (APB1PERIPH_BASE + 0x0C00UL)
- #define TIM6_BASE (APB1PERIPH_BASE + 0x1000UL)
- #define TIM7_BASE (APB1PERIPH_BASE + 0x1400UL)
- #define RTC_BASE (APB1PERIPH_BASE + 0x2800UL)
- #define WWDG_BASE (APB1PERIPH_BASE + 0x2C00UL)
- #define IWDG_BASE (APB1PERIPH_BASE + 0x3000UL)

- #define USART2_BASE (APB1PERIPH_BASE + 0x4400UL)
- #define USART3_BASE (APB1PERIPH_BASE + 0x4800UL)

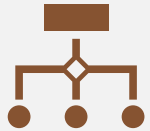
Timer Register Address map

```
typedef struct
{
  __IO uint32_t CR1;        /*!< TIM control register 1,                Address offset: 0x00 */
  __IO uint32_t CR2;        /*!< TIM control register 2,                Address offset: 0x04 */
  __IO uint32_t SMCR;       /*!< TIM slave mode control register,       Address offset: 0x08 */
  __IO uint32_t DIER;       /*!< TIM DMA/interrupt enable register,     Address offset: 0x0C */
  __IO uint32_t SR;         /*!< TIM status register,                   Address offset: 0x10 */
  __IO uint32_t EGR;        /*!< TIM event generation register,         Address offset: 0x14 */
  __IO uint32_t CCMR1;      /*!< TIM capture/compare mode register 1,   Address offset: 0x18 */
  __IO uint32_t CCMR2;      /*!< TIM capture/compare mode register 2,   Address offset: 0x1C */
  __IO uint32_t CCER;       /*!< TIM capture/compare enable register,   Address offset: 0x20 */
  __IO uint32_t CNT;        /*!< TIM counter register,                  Address offset: 0x24 */
  __IO uint32_t PSC;        /*!< TIM prescaler,                         Address offset: 0x28 */
  __IO uint32_t ARR;        /*!< TIM auto-reload register,              Address offset: 0x2C */
  __IO uint32_t RCR;        /*!< TIM repetition counter register,       Address offset: 0x30 */
  __IO uint32_t CCR1;       /*!< TIM capture/compare register 1,        Address offset: 0x34 */
  __IO uint32_t CCR2;       /*!< TIM capture/compare register 2,        Address offset: 0x38 */
  __IO uint32_t CCR3;       /*!< TIM capture/compare register 3,        Address offset: 0x3C */
  __IO uint32_t CCR4;       /*!< TIM capture/compare register 4,        Address offset: 0x40 */
  __IO uint32_t BDTR;       /*!< TIM break and dead-time register,      Address offset: 0x44 */
  __IO uint32_t DCR;        /*!< TIM DMA control register,              Address offset: 0x48 */
  __IO uint32_t DMAR;       /*!< TIM DMA address for full transfer,     Address offset: 0x4C */
  __IO uint32_t OR1;        /*!< TIM option register 1,                 Address offset: 0x50 */
  __IO uint32_t CCMR3;      /*!< TIM capture/compare mode register 3,   Address offset: 0x54 */
  __IO uint32_t CCR5;       /*!< TIM capture/compare register5,         Address offset: 0x58 */
  __IO uint32_t CCR6;       /*!< TIM capture/compare register6,         Address offset: 0x5C */
  __IO uint32_t OR2;        /*!< TIM option register 2,                 Address offset: 0x60 */
  __IO uint32_t OR3;        /*!< TIM option register 3,                 Address offset: 0x64 */
} TIM_TypeDef;
```

Example

Timer2 control register 2 Address

TIM2_BASE + 0x04 = 0x40000004U



Timer3 control register 2 Address

TIM3_BASE + 0x04 = 0x40000404U

Bus Address Map

Bus	Boundary address	Size (bytes)	Peripheral
AHB3	0xA000 1000 - 0xA000 13FF	1 KB	QUADSPI
	0xA000 0000 - 0xA000 0FFF	4 KB	FMC
AHB2	0x5006 0800 - 0x5006 0BFF	1 KB	RNG
	0x5004 0400 - 0x5006 07FF	129 KB	Reserved
	0x5004 0000 - 0x5004 03FF	1 KB	ADC
	0x5000 0000 - 0x5003 FFFF	16 KB	OTG_FS
	0x4800 2000 - 0x4FFF FFFF	~127 MB	Reserved
	0x4800 1C00 - 0x4800 1FFF	1 KB	GPIOH
	0x4800 1800 - 0x4800 1BFF	1 KB	GPIOG
	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF
	0x4800 1000 - 0x4800 13FF	1 KB	GPIOE
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA
-	0x4002 4400 - 0x47FF FFFF	~127 MB	Reserved
AHB1	0x4002 4000 - 0x4002 43FF	1 KB	TSC
	0x4002 3400 - 0x4002 3FFF	1 KB	Reserved
	0x4002 3000 - 0x4002 33FF	1 KB	CRC
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH registers
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved
	0x4002 1000 - 0x4002 13FF	1 KB	RCC
	0x4002 0800 - 0x4002 0FFF	2 KB	Reserved
	0x4002 0400 - 0x4002 07FF	1 KB	DMA2
	0x4002 0000 - 0x4002 03FF	1 KB	DMA1

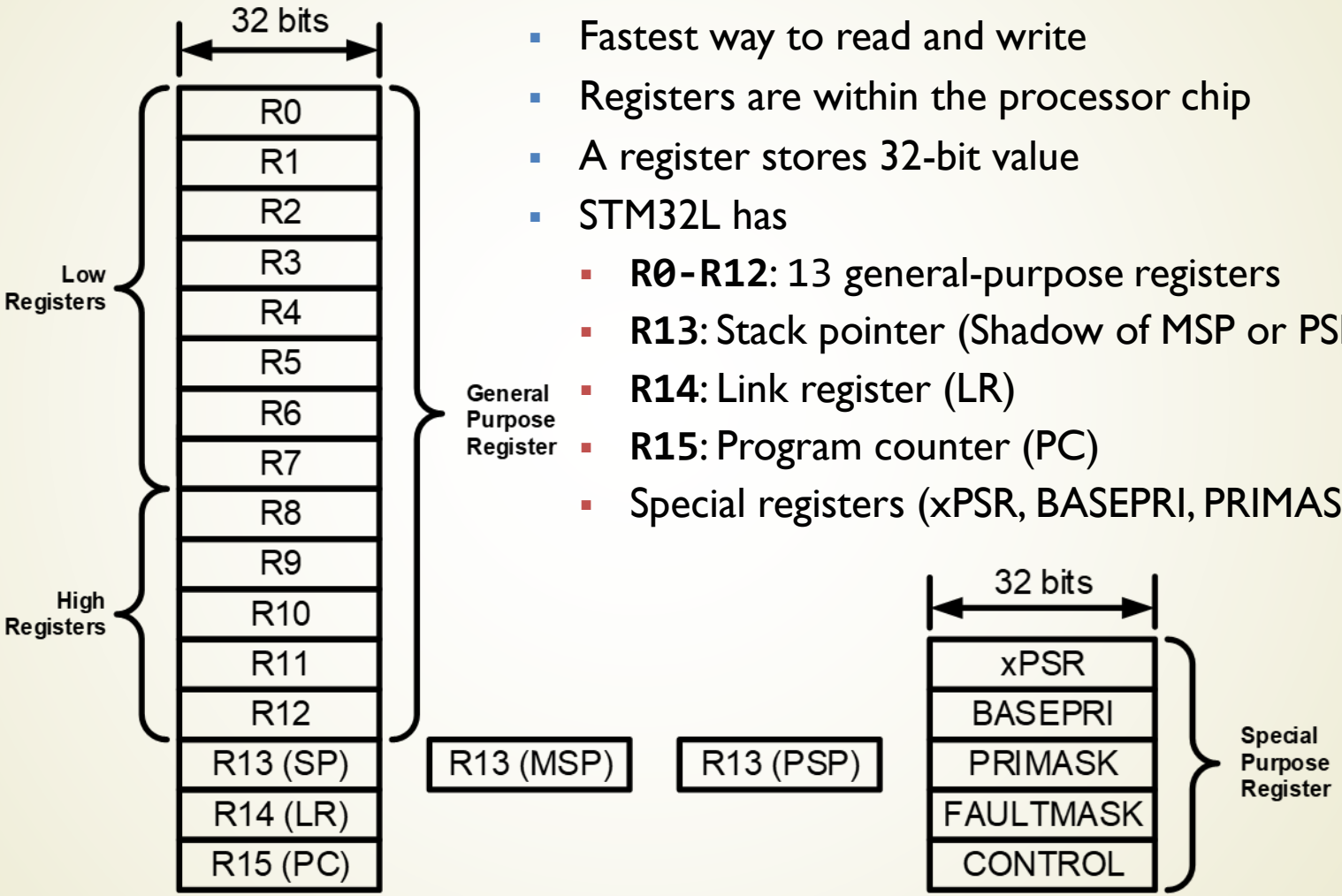
Bus Address Map

Bus	Boundary address	Size (bytes)	Peripheral
APB1	0x4000 9800 - 0x4000 FFFF	26 KB	Reserved
	0x4000 9400 - 0x4000 97FF	1 KB	LPTIM2
	0x4000 8C00 - 0x4000 93FF	2 KB	Reserved
	0x4000 8800 - 0x4000 8BFF	1 KB	SWPMI1
	0x4000 8400 - 0x4000 87FF	1 KB	Reserved
	0x4000 8000 - 0x4000 83FF	1 KB	LPUART1
	0x4000 7C00 - 0x4000 7FFF	1 KB	LPTIM1
	0x4000 7800 - 0x4000 7BFF	1 KB	OPAMP
	0x4000 7400 - 0x4000 77FF	1 KB	DAC1
	0x4000 7000 - 0x4000 73FF	1 KB	PWR
	0x4000 6800 - 0x4000 6FFF	1 KB	Reserved
	0x4000 6400 - 0x4000 67FF	1 KB	CAN1
	0x4000 6000 - 0x4000 63FF	1 KB	Reserved
	0x4000 5C00- 0x4000 5FFF	1 KB	I2C3
	0x4000 5800 - 0x4000 5BFF	1 KB	I2C2
	0x4000 5400 - 0x4000 57FF	1 KB	I2C1
	0x4000 5000 - 0x4000 53FF	1 KB	UART5
	0x4000 4C00 - 0x4000 4FFF	1 KB	UART4
	0x4000 4800 - 0x4000 4BFF	1 KB	USART3
	0x4000 4400 - 0x4000 47FF	1 KB	USART2

Bus Address Map

Bus	Boundary address	Size (bytes)	Peripheral
APB2	0x4001 6400 - 0x4001 FFFF	39 KB	Reserved
	0x4001 6000 - 0x4000 63FF	1 KB	DFSDM1
	0x4001 5C00 - 0x4000 5FFF	1 KB	Reserved
	0x4001 5800 - 0x4000 5BFF	1 KB	SAI2
	0x4001 5400 - 0x4000 57FF	1 KB	SAI1
	0x4001 4C00 - 0x4000 53FF	2 KB	Reserved
	0x4001 4800 - 0x4001 4BFF	1 KB	TIM17
	0x4001 4400 - 0x4001 47FF	1 KB	TIM16
	0x4001 4000 - 0x4001 43FF	1 KB	TIM15
APB2	0x4001 3C00 - 0x4001 3FFF	1 KB	Reserved
	0x4001 3800 - 0x4001 3BFF	1 KB	USART1
	0x4001 3400 - 0x4001 37FF	1 KB	TIM8
	0x4001 3000 - 0x4001 33FF	1 KB	SPI1
	0x4001 2C00 - 0x4001 2FFF	1 KB	TIM1
	0x4001 2800 - 0x4001 2BFF	1 KB	SDMMC1
	0x4001 2000 - 0x4001 27FF	2 KB	Reserved
	0x4001 1C00 - 0x4001 1FFF	1 KB	FIREWALL
	0x4001 0800- 0x4001 1BFF	5 KB	Reserved
	0x4001 0400 - 0x4001 07FF	1 KB	EXTI
	0x4001 0200 - 0x4001 03FF	1 KB	COMP
	0x4001 0030 - 0x4001 01FF		VREFBUF
	0x4001 0000 - 0x4001 002F		SYSCFG

Processor Registers



- Fastest way to read and write
- Registers are within the processor chip
- A register stores 32-bit value
- STM32L has
  - **R0-R12**: 13 general-purpose registers
  - **R13**: Stack pointer (Shadow of MSP or PSP)
  - **R14**: Link register (LR)
  - **R15**: Program counter (PC)
  - Special registers (xPSR, BASEPRI, PRIMASK, etc)

Register Keyword

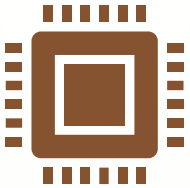
- Register keyword in C/C++
  - Indication to compiler to store in processor registers
  - For faster execution
- Registers are used for faster execution – why ?
  - No memory operation is involved

Peripheral registers

- Three types of registers
  - Read Only
  - Write Only
  - Read/Write

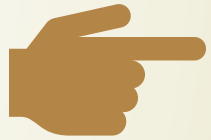
Peripheral Registers

Each peripherals have many registers



Don't have to program all the registers

Program only those registers that is needed.



For example: SysTick has 4 registers

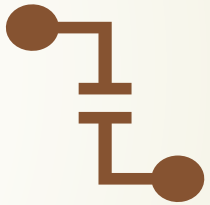
Only 3 registers are used in most cases

Peripheral Registers

Link Register The Link Register (LR) is register R14.

Stores the return information for subroutines, function calls, and exceptions

On reset, the processor sets the LR value to 0xFFFFFFFF.

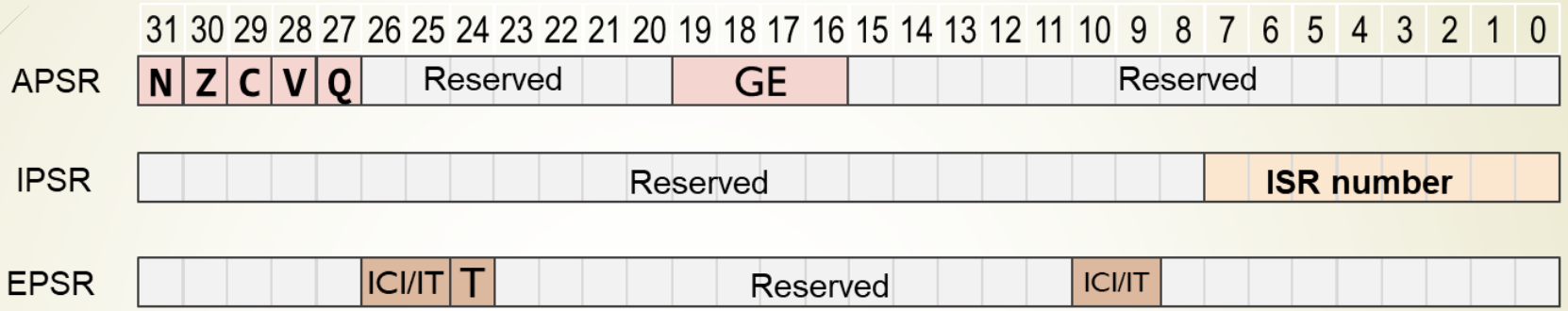


Program Counter The Program Counter (PC) is register R15.

Contains the current program address.

On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004

PSR combines: Application PSR (APSR), Interrupt PSR (IPSR), Execution PSR (EPSR)



Combine them together into one register (PSR)

Note:

- GE flags are only available on Cortex-M4 and M7
- Use PSR in code if needed

Program Status Registers

Flags	
N (Negative)	This flag is set when Result is negative
Z (Zero)	This flag is set when result is zero
C (Carry)	Carry flag is set in unsigned addition and subtraction
V (Overflow)	Overflow is set when doing signed addition or subtraction
Q (Negative)	Saturation flag is set when SSAT and USAT instruction causes saturation

Program Status Register

Fault Mask Register

- FAULTMASK register prevents activation of all exceptions except for Non-Maskable Interrupt (NMI)

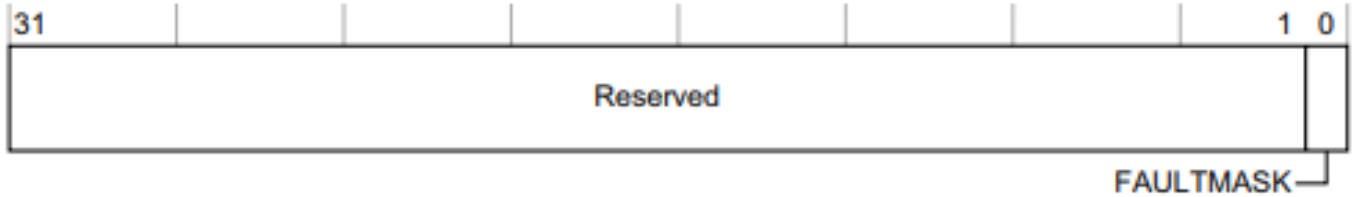


Table 2-8 FAULTMASK register bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	FAULTMASK	0 = no effect 1 = prevents the activation of all exceptions except for NMI.

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

Priority Mask Register

- PRIMASK register prevents activation of all exceptions with configurable priority.

Table 2-7 PRIMASK register bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	PRIMASK	0 = no effect 1 = prevents the activation of all exceptions with configurable priority.

Base Priority Mask Register

- BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value

Table 2-9 BASEPRI register bit assignments

Bits	Name	Function
[31:8]	-	Reserved
[7:0]	BASEPRI[a]	Priority mask bits: 0x00 = no effect Nonzero = defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.

Stack pointer

- The Stack Pointer (SP) is register R13.
  - In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use
  - 0 = Main Stack Pointer (MSP). This is the reset value
  - 1 = Process Stack Pointer (PSP).

Code Execution

Assembly Code

```
        MOVS r1, #0
        MOVS r0, #0
        B    check
loop    ADD  r1, r1, r0
        ADDS r0, r0, #1
check   CMP  r0, #10
        BLT  loop
self    B    self
```

Assembler

Machine Code

```
0010000100000000
0010000000000000
1110000000000001
0100010000000001
0001110001000000
0010100000001010
1101110011111011
1011111100000000
1110011111111110
```

- Assembly language
  - Textual representation of instructions
  - Assembler to convert to Machine code
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

C Code

```
int main(void){
    int a = 2;
    int b = 1;
    int c;
    c = a + b;
    return 0;
}
```

compiler

Assembly Code

```
MOVS r1, #0x00      ; int a = 2
MOVS r2, #0x01      ; int b = 1
ADDS r3, r1, r2     ; c = a + b
MOVS r0, 0x00       ; set return value
BX lr               ; return
```

assembler

Machine Code

```
0010000100000000
0010001000000001
0001100010001011
0010000000000000
0100011101110000
```

In Binary

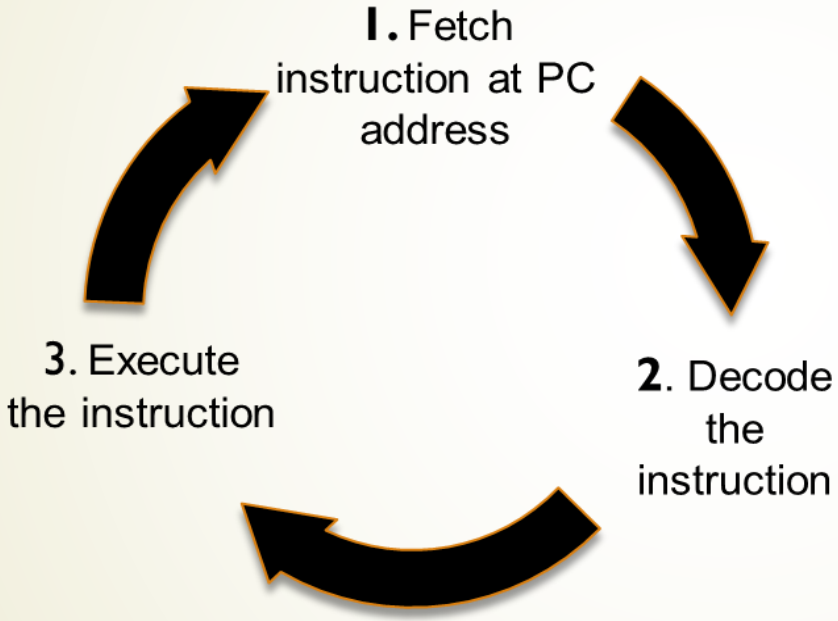
```
2100
2201
188B
2000
4770
```

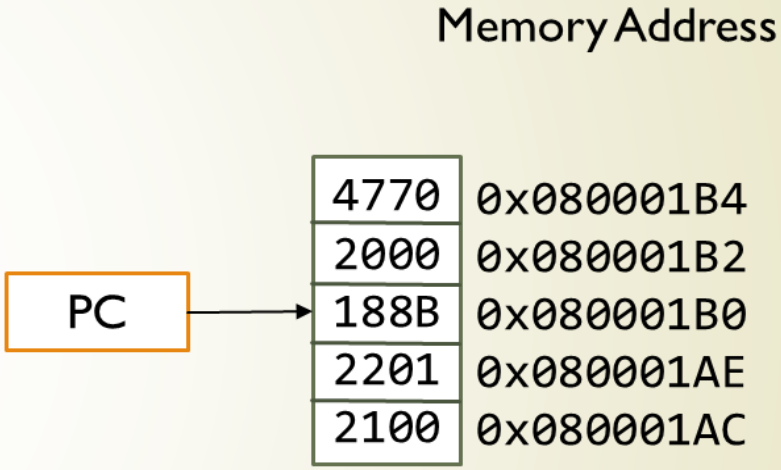
In Hex

```
; MOVS    r1, #0x02
; MOVS    r2, #0x01
; ADDS    r3, r1, r2
; MOVS    r0, #0x00
; BX      lr
```


Code Execution

- Program Counter (PC) holds the memory address of the next instruction to be fetched from the memory.



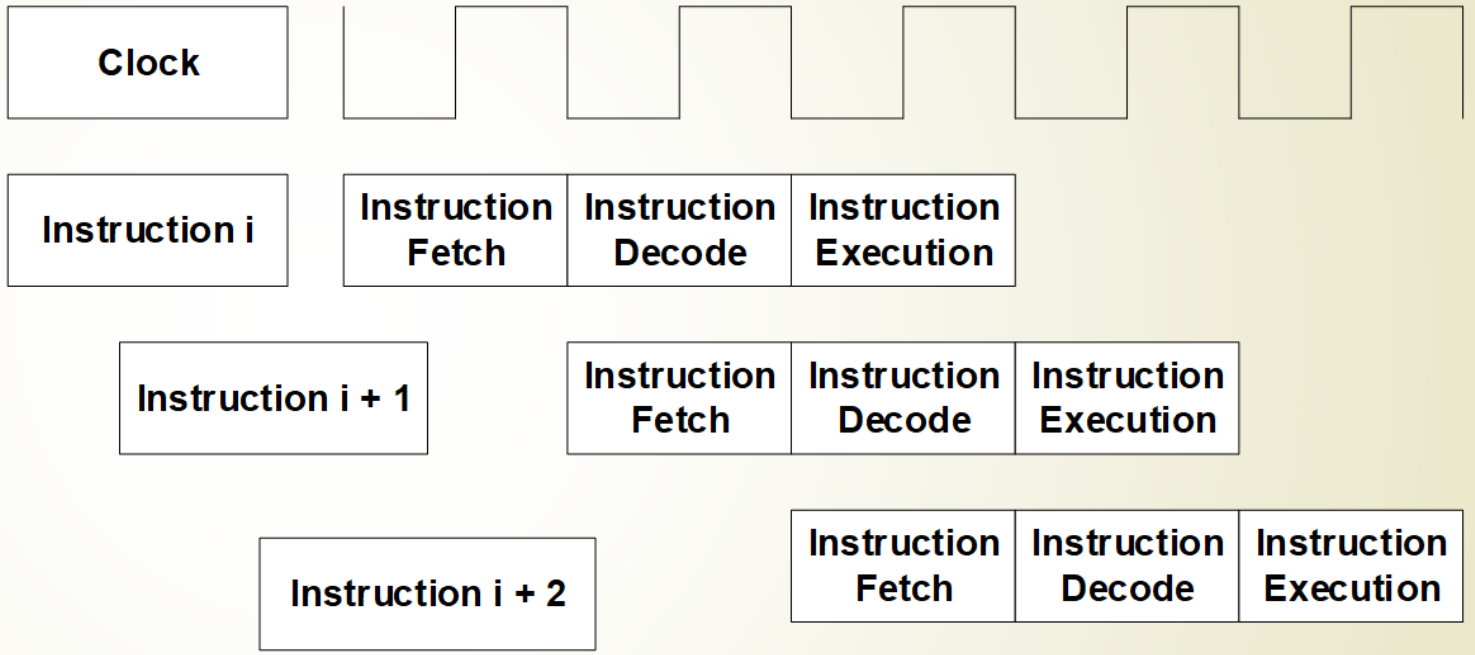


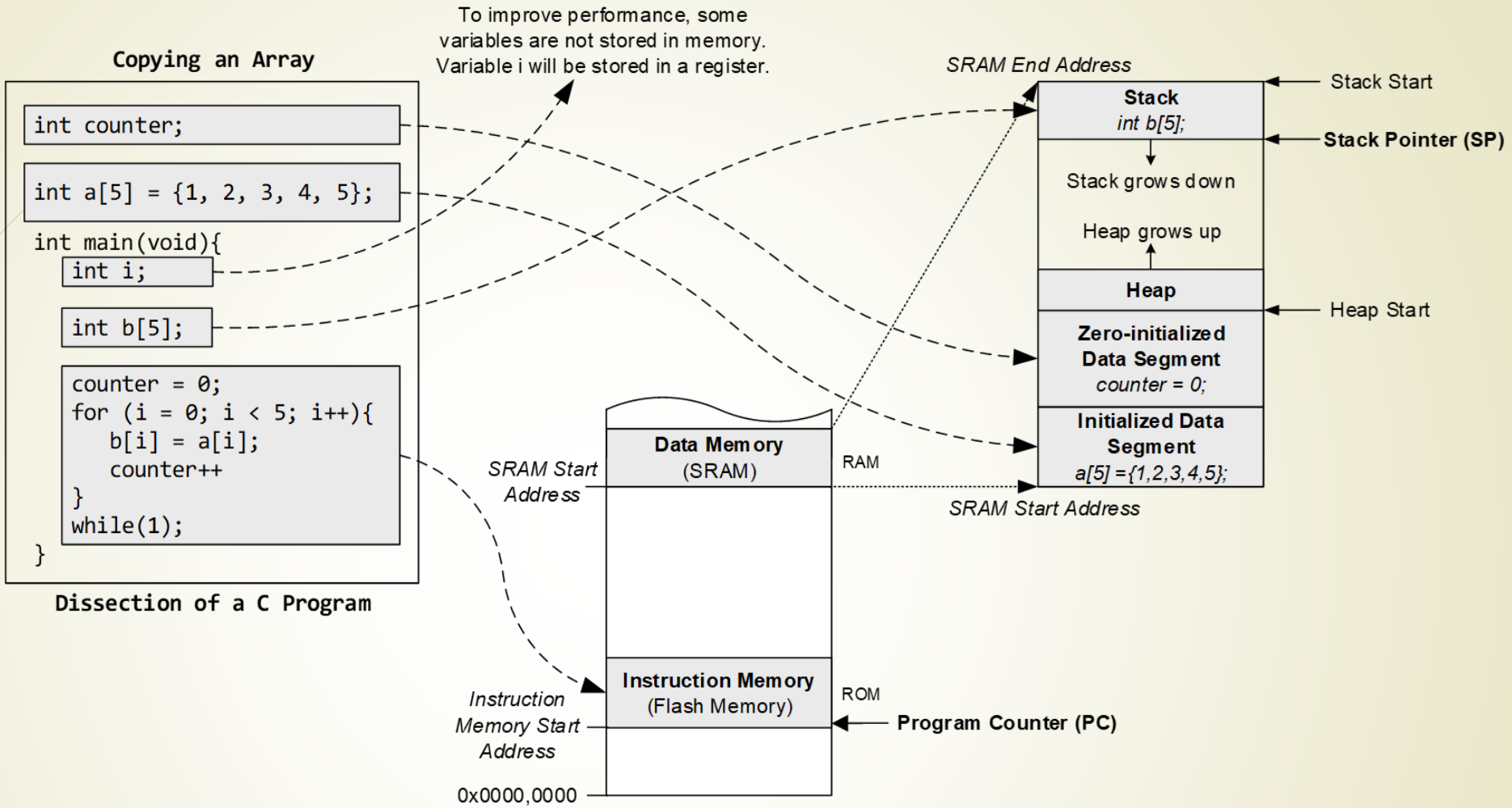
PC = 0x080001B0
Instruction = 188B or
2000188B or 8B180020

Program Execution

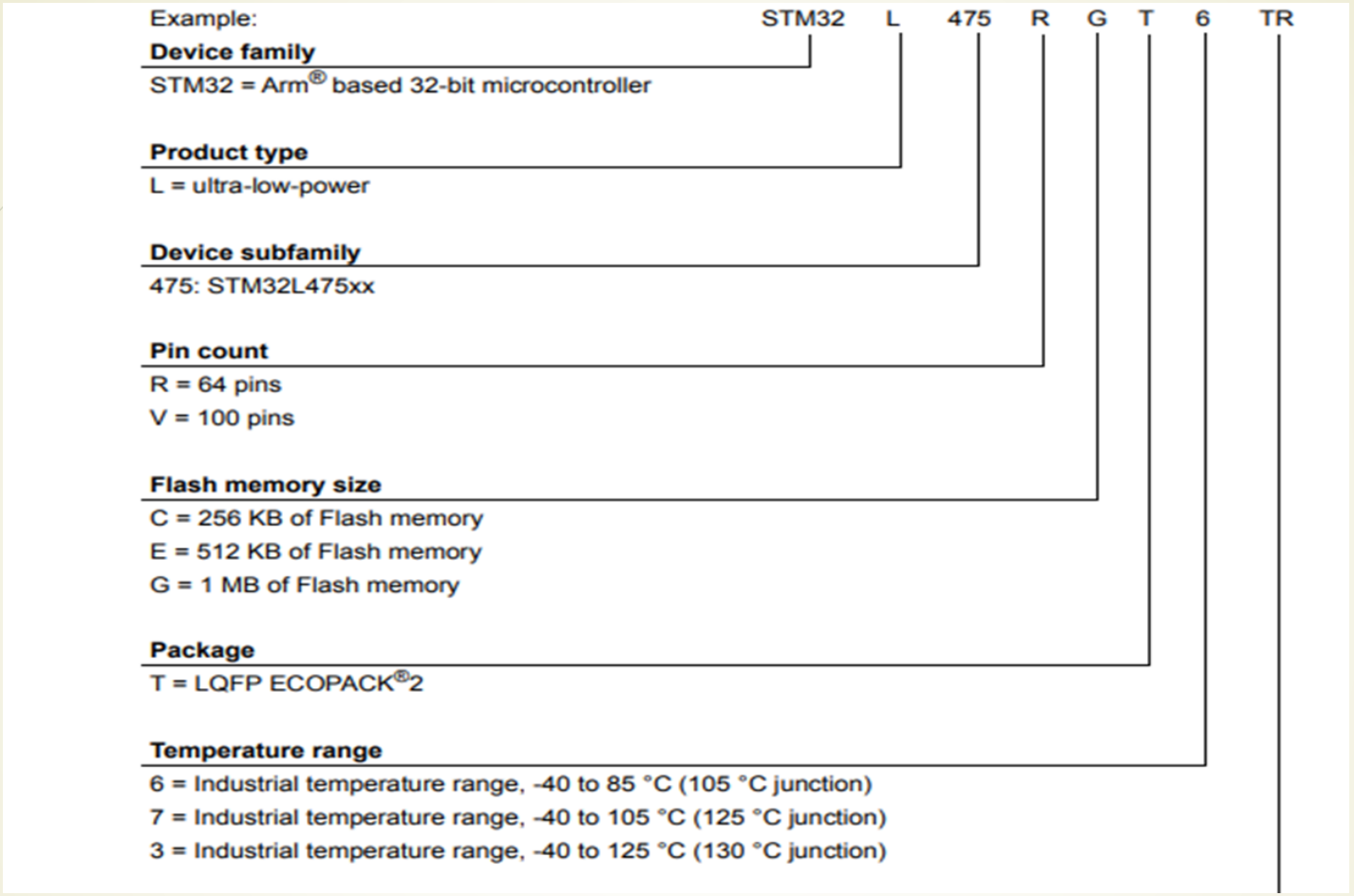
Pipeline

- Pipelining allows hardware resources to be fully utilized
- One 32-bit instruction or two 16-bit instructions can be fetched.





Code loading in Memory



Processor Packaging

Next Lesson Topic

Instruction Sets

Assembly instruction

Instruction format

Arithmetic and Logical Instructions

Conditional logic