CS301
Embedded System and Microcomputer Principle

Lecture 9: DMA&Pipeline

2024 Fall

baiyh@sustech.edu.cn

Outline

- **DMA**
- Bit Banding
- ARM Architecture - Pipeline

I/O Interface

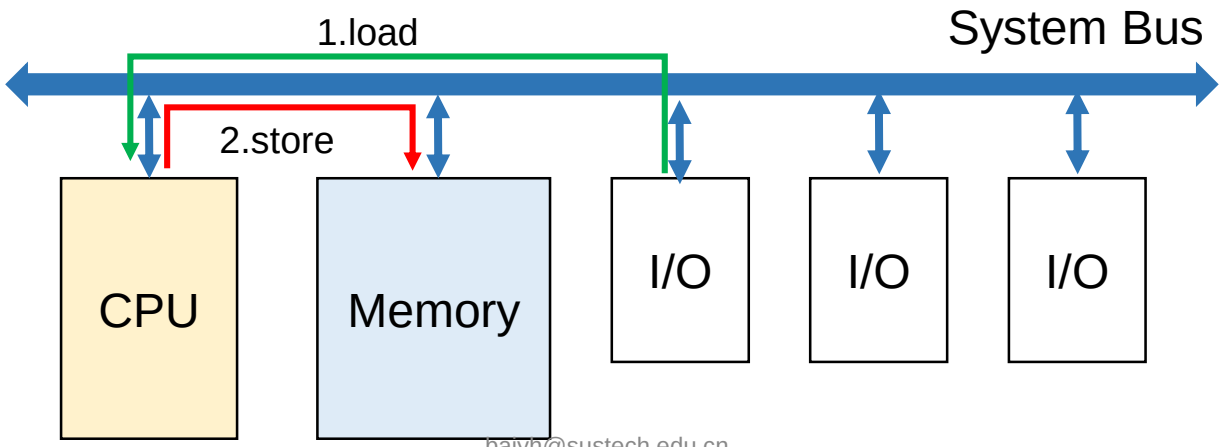
- Three I/O control methods
  - Polling
  - Interrupt
  - Direct Memory Access (DMA)

baiyh@sustech.edu.cn

Data Transfer Schemes

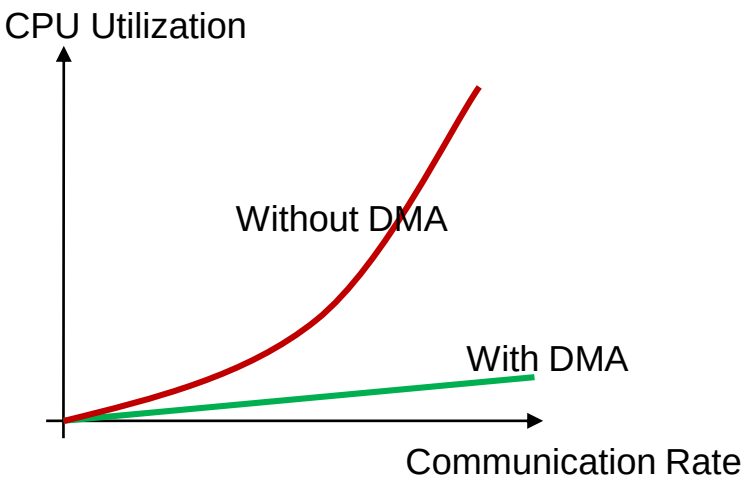
- In a computer, the data transfer happens between any of these combinations
  - CPU and memory
  - CPU and I/O devices
  - I/O devices and memory
- How to move data between I/O and Memory?
  1. Load from peripheral (memory mapped IO)
  2. Store into memory

```
LDA r0, [r1]
STA r0, [r2]
```



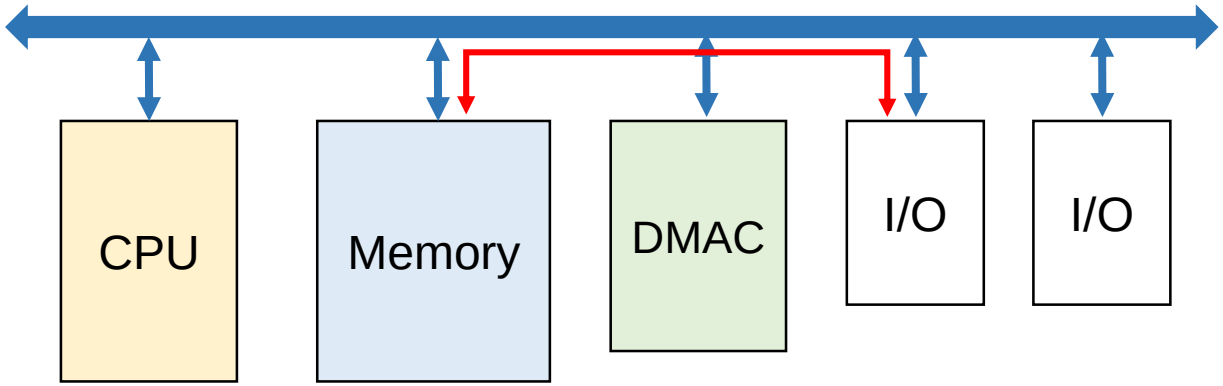
Why Need DMA?

- If transferring data without the use of DMA
  - Each pure data transfer requires 2 cycles (load + store)
  - waste of CPU instructions: Instructions are needed to increment memory address and keeping track of how many bytes are moved.



Direct Memory Access (DMA)

- To transfer large blocks of data at high speed, an alternative approach is used, called DMA.
- Blocks of data are transferred between an external device and the main memory, or between memory to memory, without continuous intervention by the processor.
- DMAC is the control unit for DMA transfer, it is just another peripheral whose job is moving data.



DMA Transfer Mode

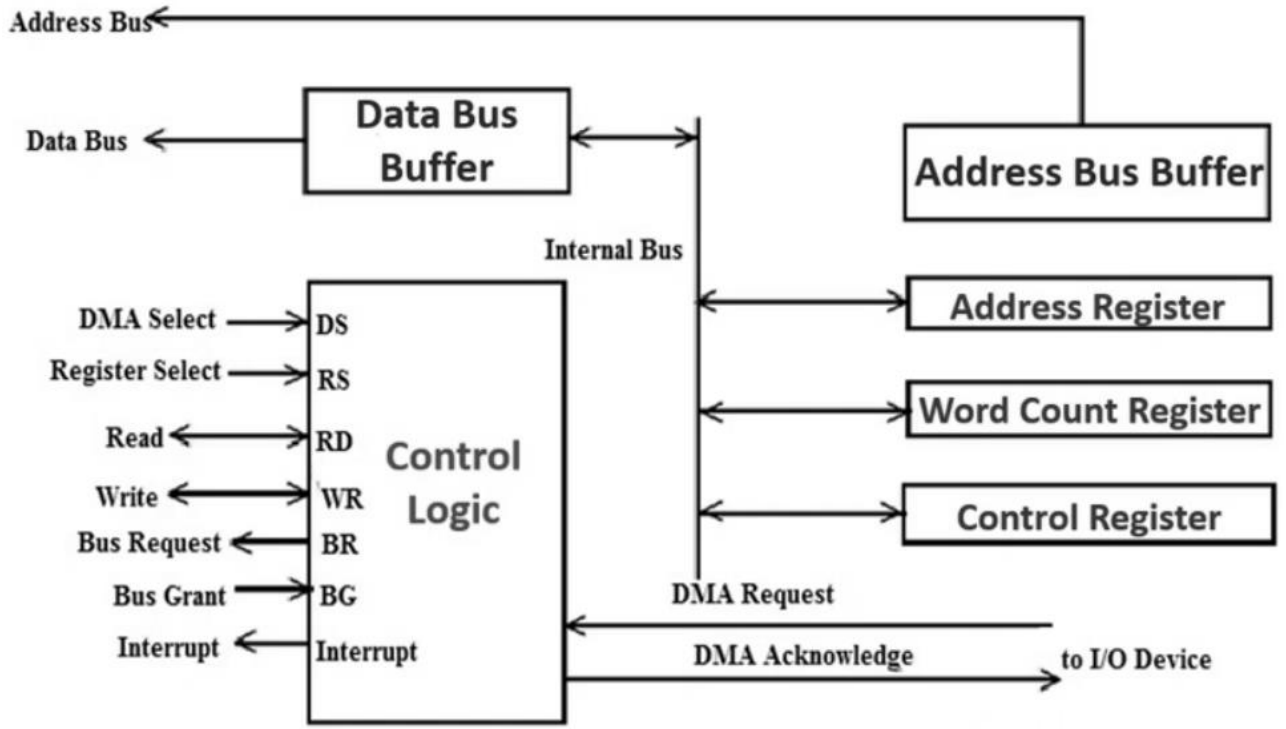
- Burst Mode
  - An entire block of data is transferred in one continuous operation, making it highly efficient.
  - The CPU remains inactive for relatively long periods of time during data transfer.
- Interleaving Mode
  - DMA transfers data only when there’s no conflict with CPU for system bus or memory access, ensuring no interference
- Cycle Stealing Mode
  - DMA controller takes control, transfers one word of data, releases control back to the CPU, and then repeats this process. This allows the CPU to continue executing its instructions between data transfers

CPU Stall

- When DMAC takes over the bus access, or when there's a conflict accessing memory, CPU is stalled
- Stall is not an interrupt, as CPU does not switch context
- However, CPU still might execute instruction taking from cache if there's no cache miss

DMA Controller (DMAC)

- Three registers:
  - **Address register**: contains the address to specify the desired location in memory
  - **Word count register**: contains the number of words to be transferred
  - **Control register**: specifies the transfer mode



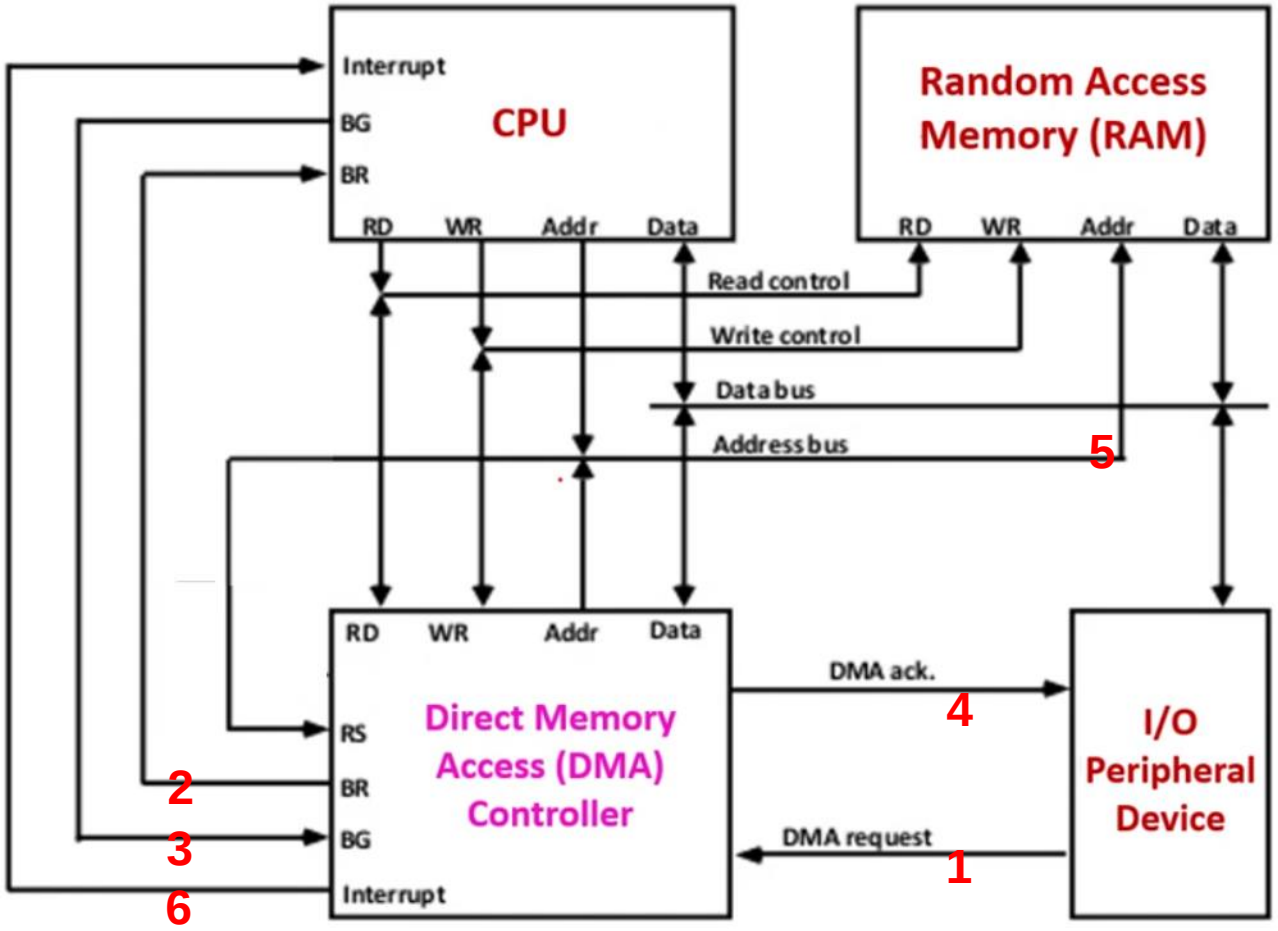
DMA Operation

- Interaction with CPU
  - CPU tells DMA controller:
    - Data transfer direction between Mem and I/O
      - Read: e.g. read a file from disk to memory
      - Write: e.g. write memory content into disk
    - I/O Device address
    - Starting address of memory block for data
    - Amount of data to be transferred
  - CPU carries on with other work
  - DMA controller deals with transfer
  - DMA controller sends interrupt when finished

DMA Operation

- Interaction with peripheral
  - When a word of data is available, the I/O device places a signal on the DMA-request wire.
  - The signal causes the DMA controller to seize the memory bus,
    - To place the desired address on the memory-address wire
    - To place a signal on the DMA-acknowledge wire
  - When the I/O device receives the DMA-acknowledge signal,
    - The transfer between external device and DMA begins

DMA Operation

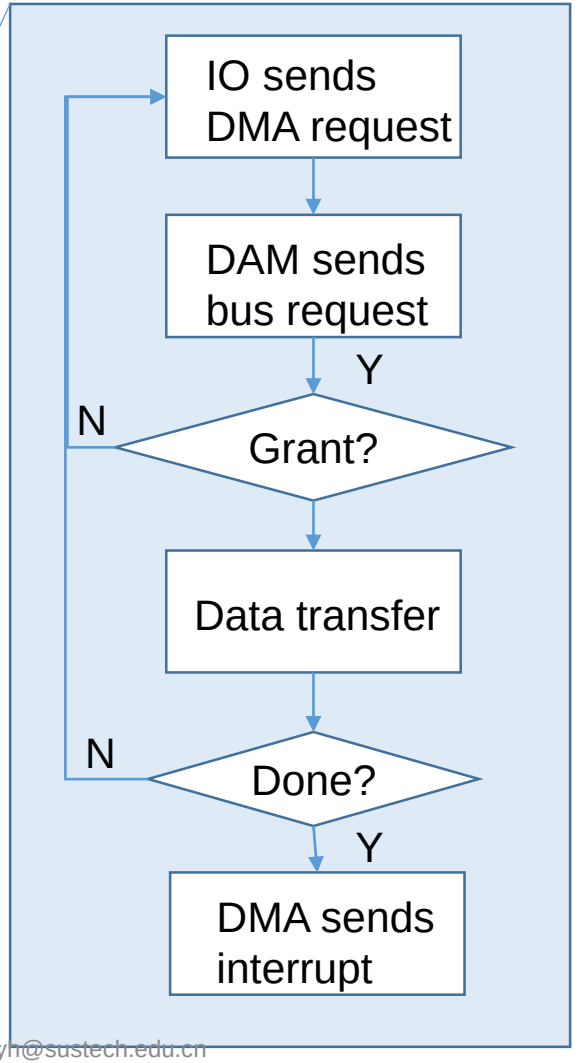


DMA Operation

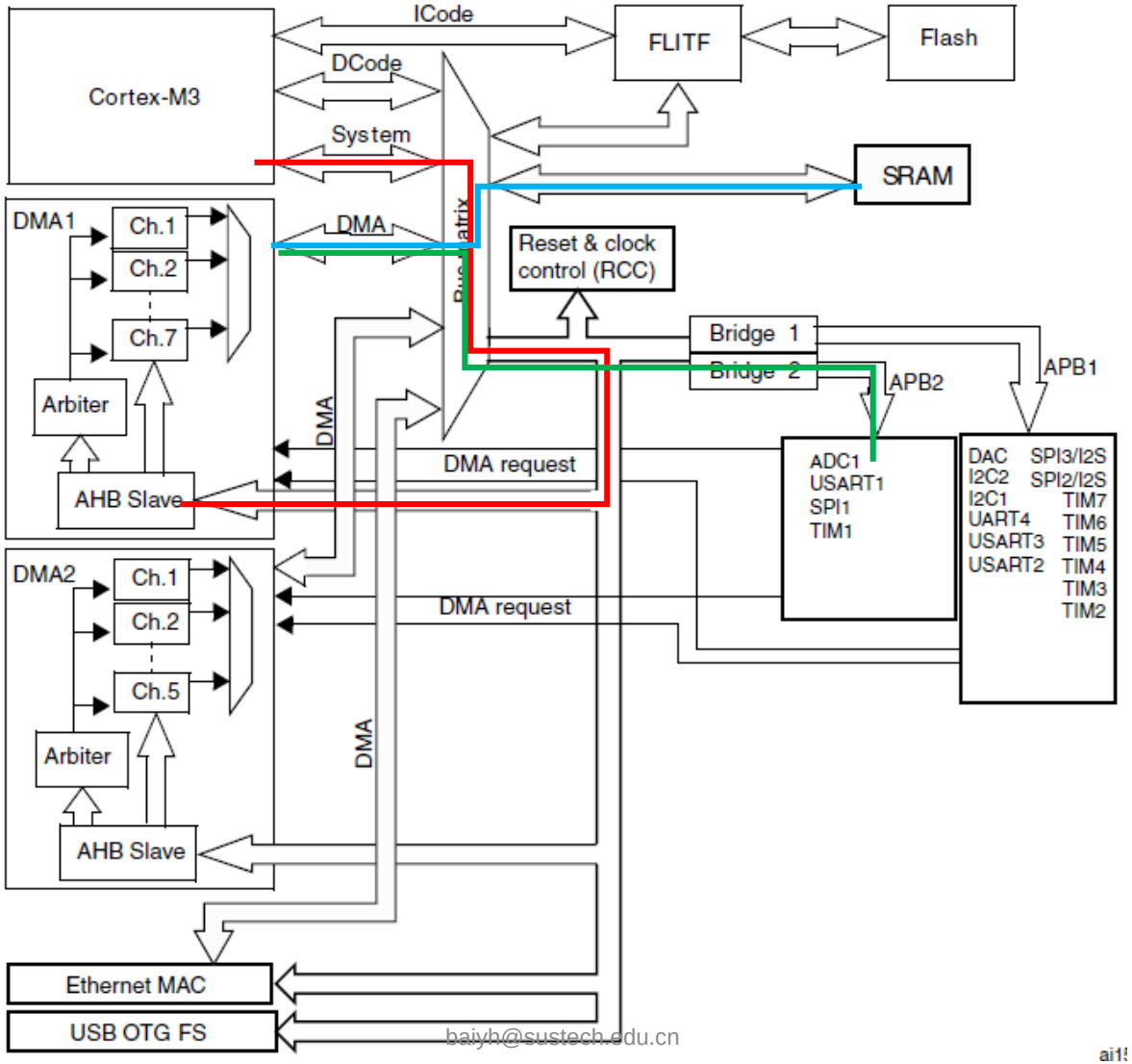
CPU tells:
Address
Word count
direction

CPU carries on with other insts
DMA handles block transfer

CPU is interrupt
Execute ISR



STM32 DMA Block Diagram



STM32 DMA Channels

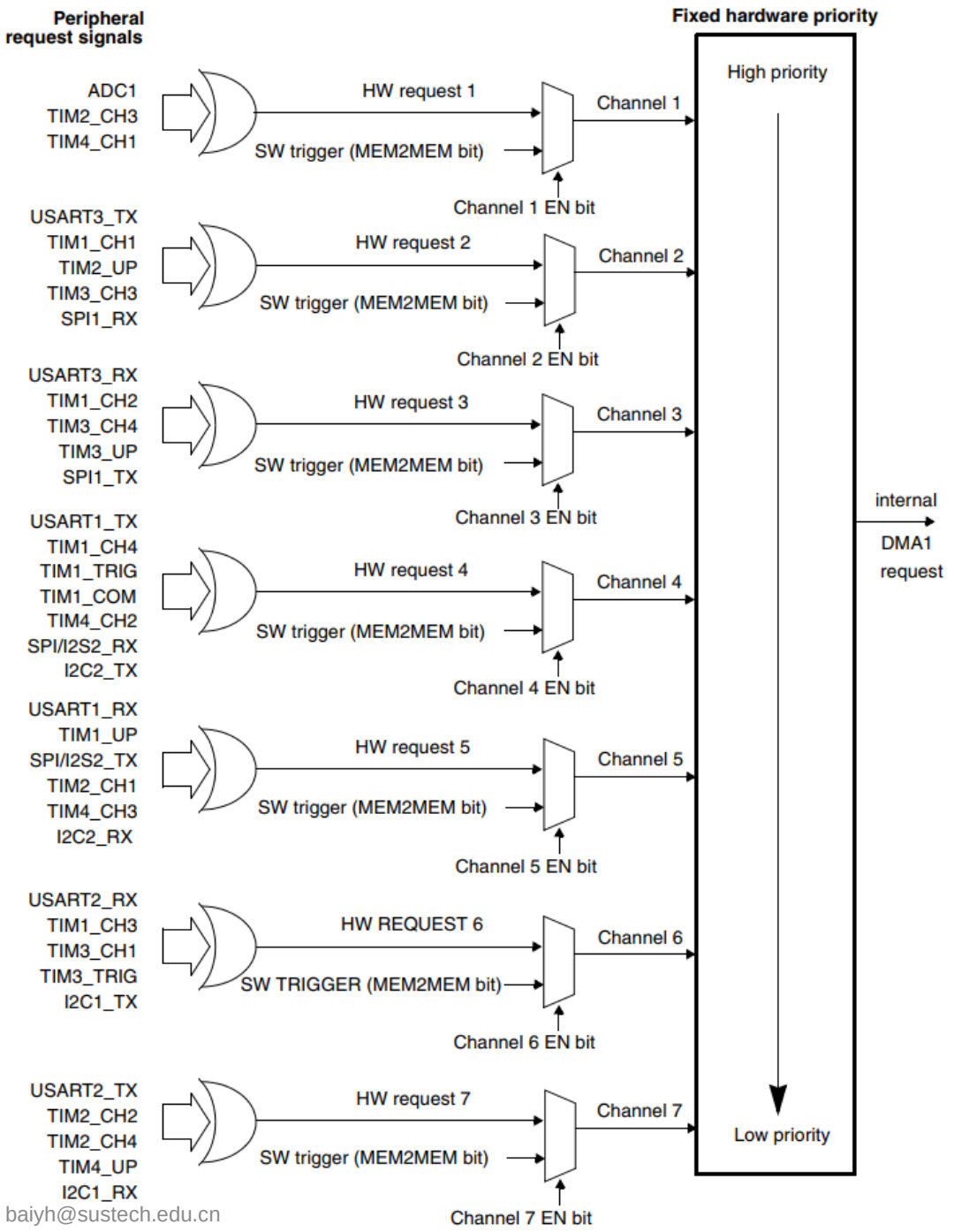
- DMA releases CPU from moving data
  - between peripherals and memory, or
  - between one peripheral and another peripheral.
- DMA uses bus matrix to allow concurrent transfers

STM32 DMA1

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI		SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I2C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP
TIM6 DAC_Ch1		TIM6_UP DAC_Ch1					
TIM7 DAC_CH2			TIM7_UP DAC_CH2				

STM32 DMA

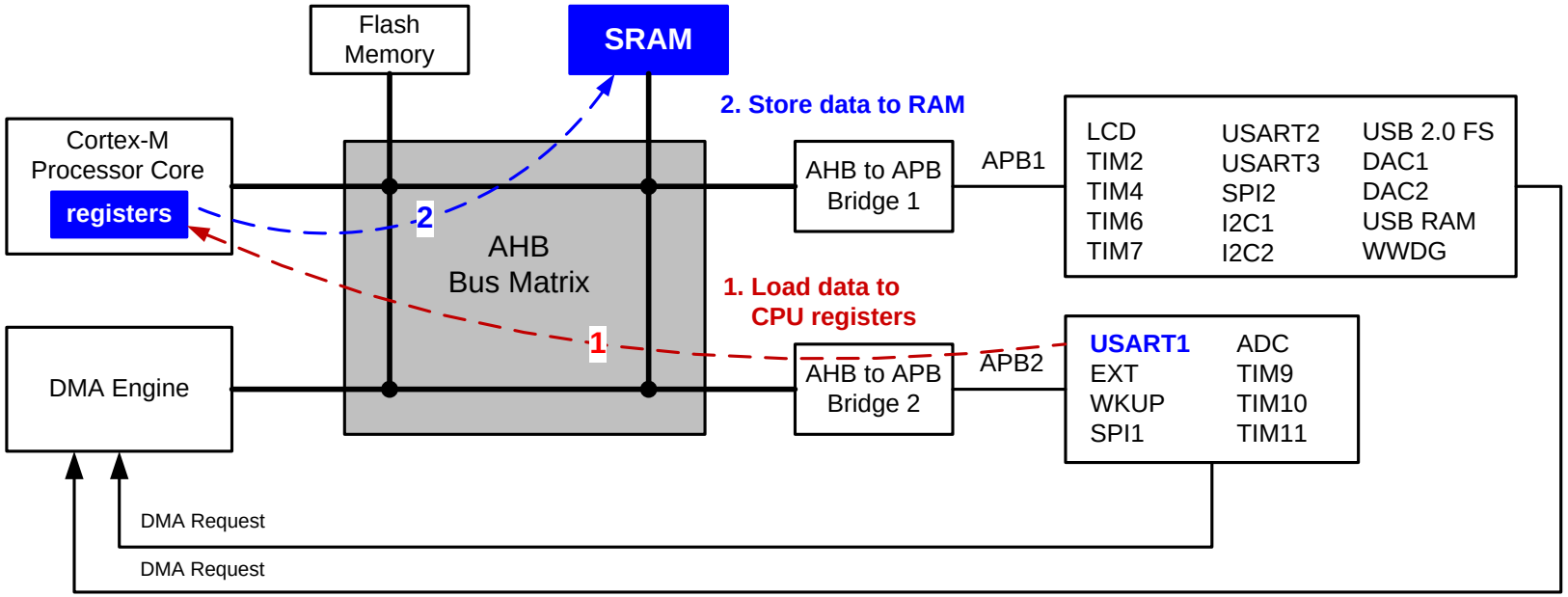
- 12 channels
  - DMA1 has 7 channels
  - DMA2 has 5 channels



baiyh@sustech.edu.cn

Programmed I/Os

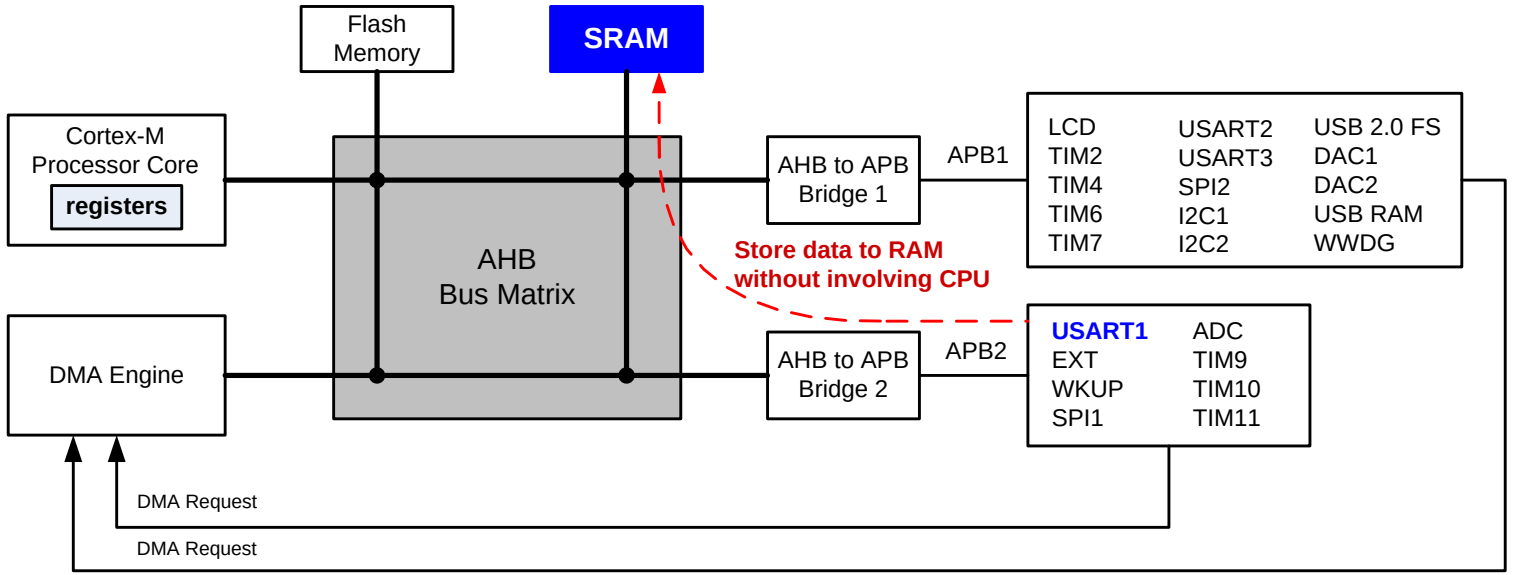
- Processor executes a lot Loads/Stores to move data
- High overhead and slow



Receiving data from USART serial port without using DMA

DMA Sets Core Free

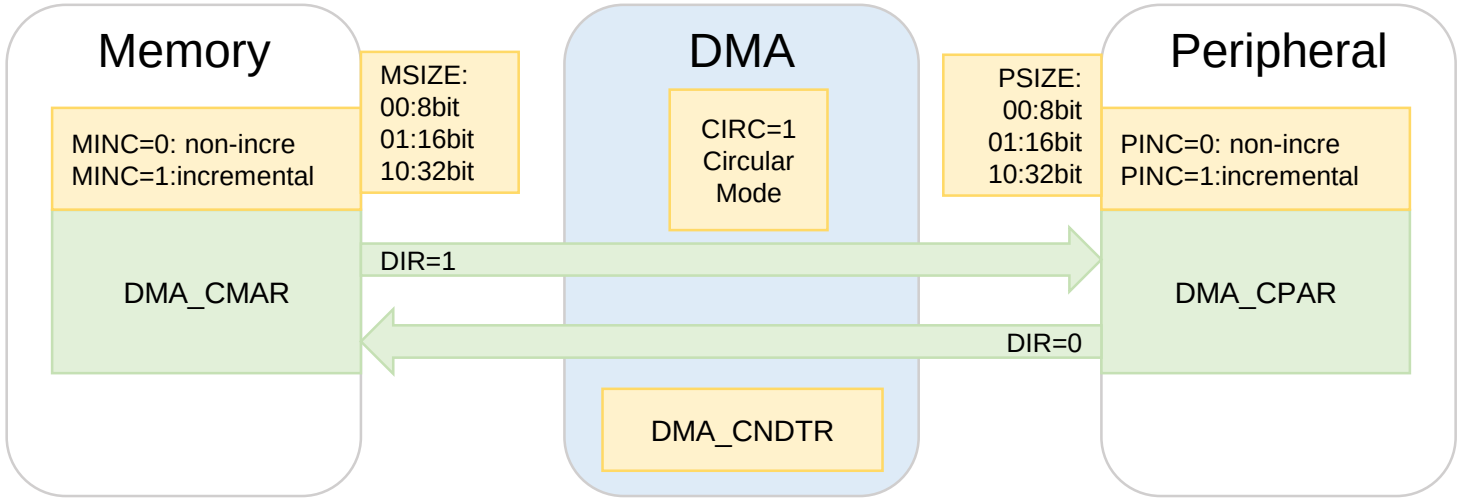
- CPU delegates reads/writes to DMA controller
- Low overhead and fast



Receiving data from USART serial port using DMA

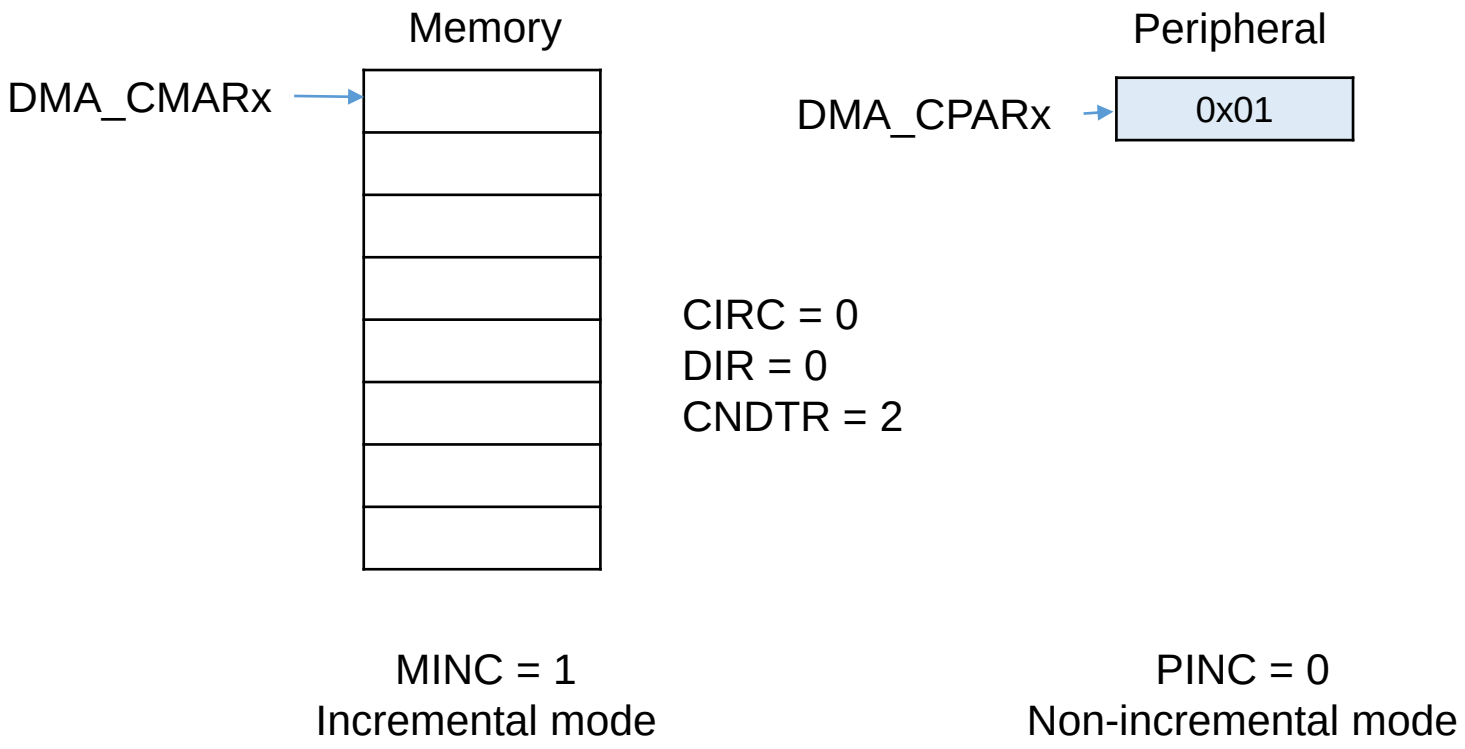
DMA Controller

- Key DMA Controller Registers
  - DMA memory address register (CMAR)
  - DMA peripheral address register (CPAR)
  - DMA number of data register (CNDTR)
  - DMA configuration register (CCR)
- DMA are often used together with interrupts



STM32 DMA Transfer

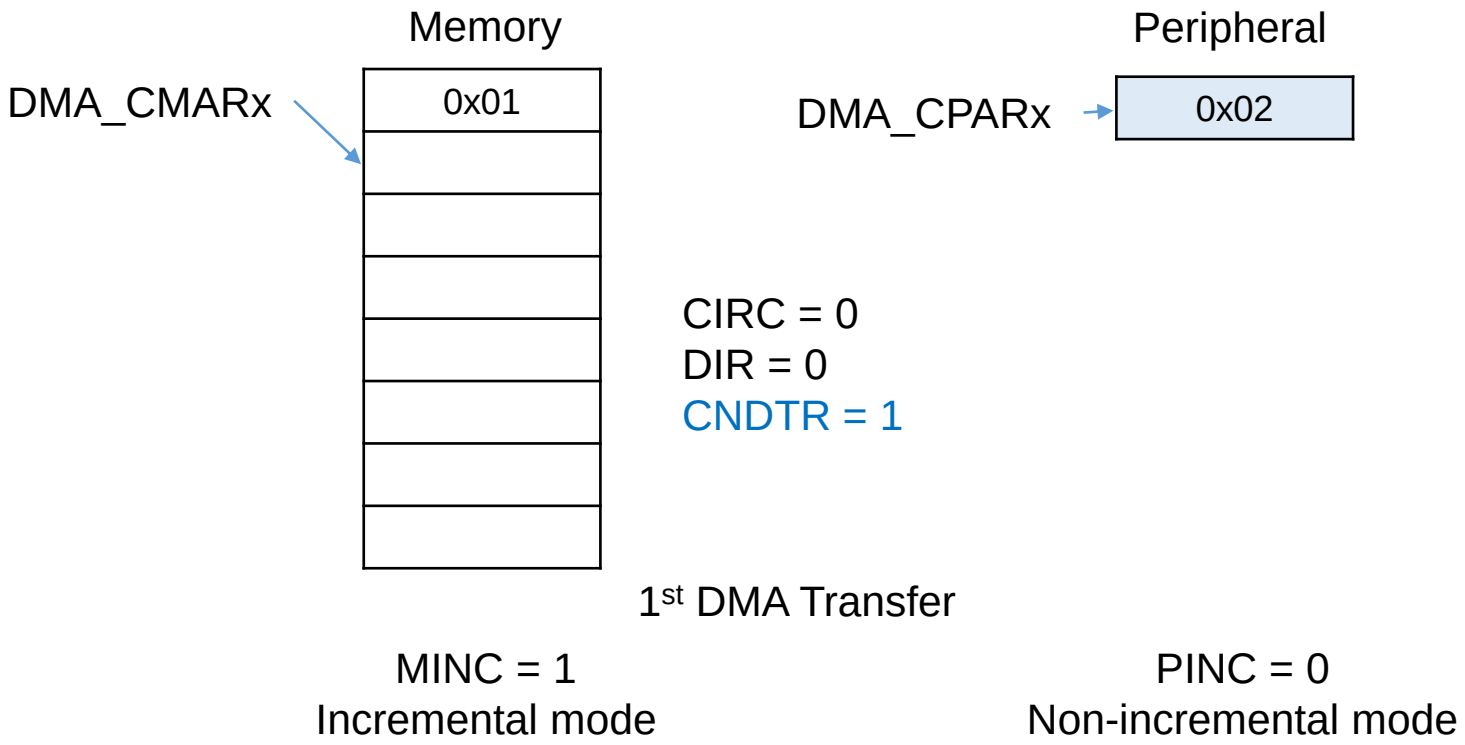
- Incremental mode in one side
  - MINC = 1, PINC = 0
  - MSIZE = 0: 8bit, PSIZE = 0: 8bit



baiyh@sustech.edu.cn

STM32 DMA Transfer

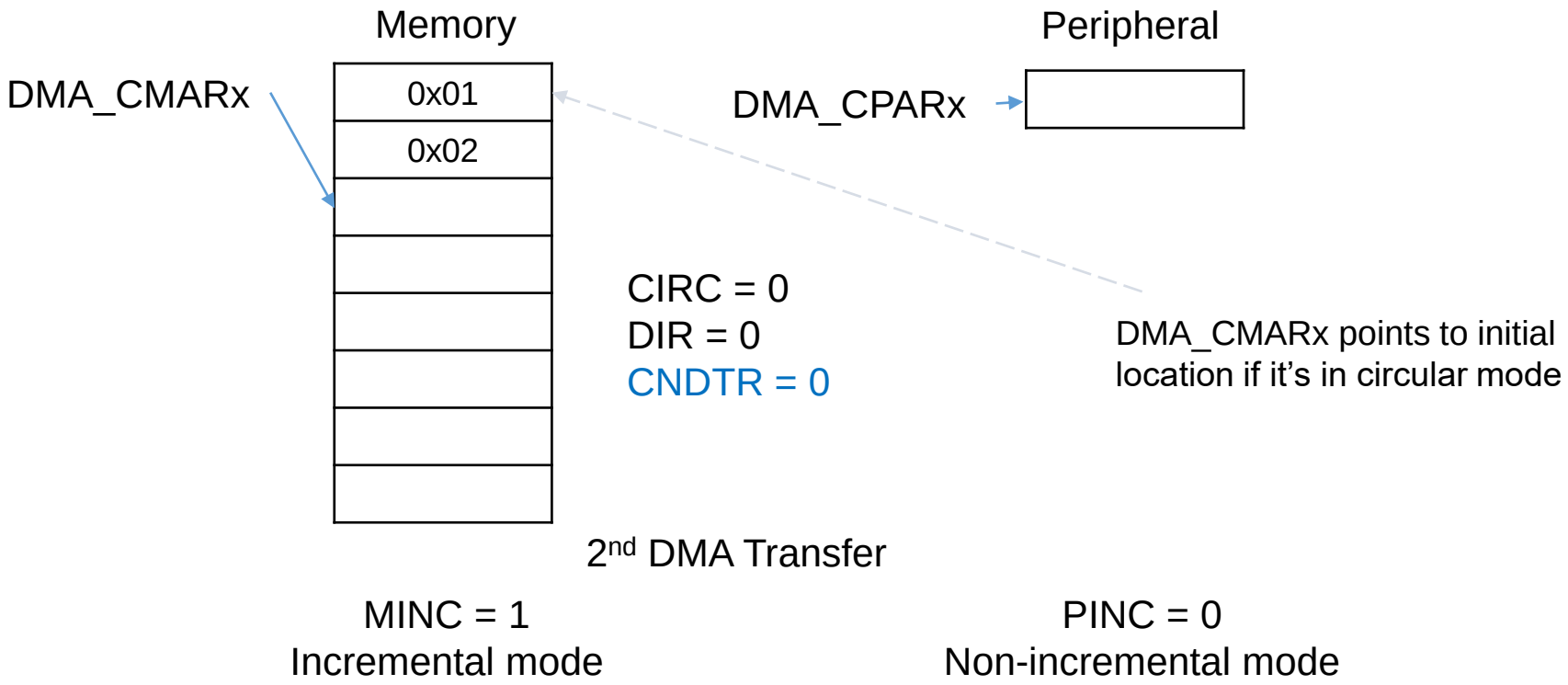
- Incremental mode in one side
  - MICN = 1, PINC = 0
  - MSIZE = 0: 8bit, PSIZE = 0: 8bit
  - DIR = 0: Transfer data from peripheral to Memory



baiyh@sustech.edu.cn

STM32 DMA Transfer

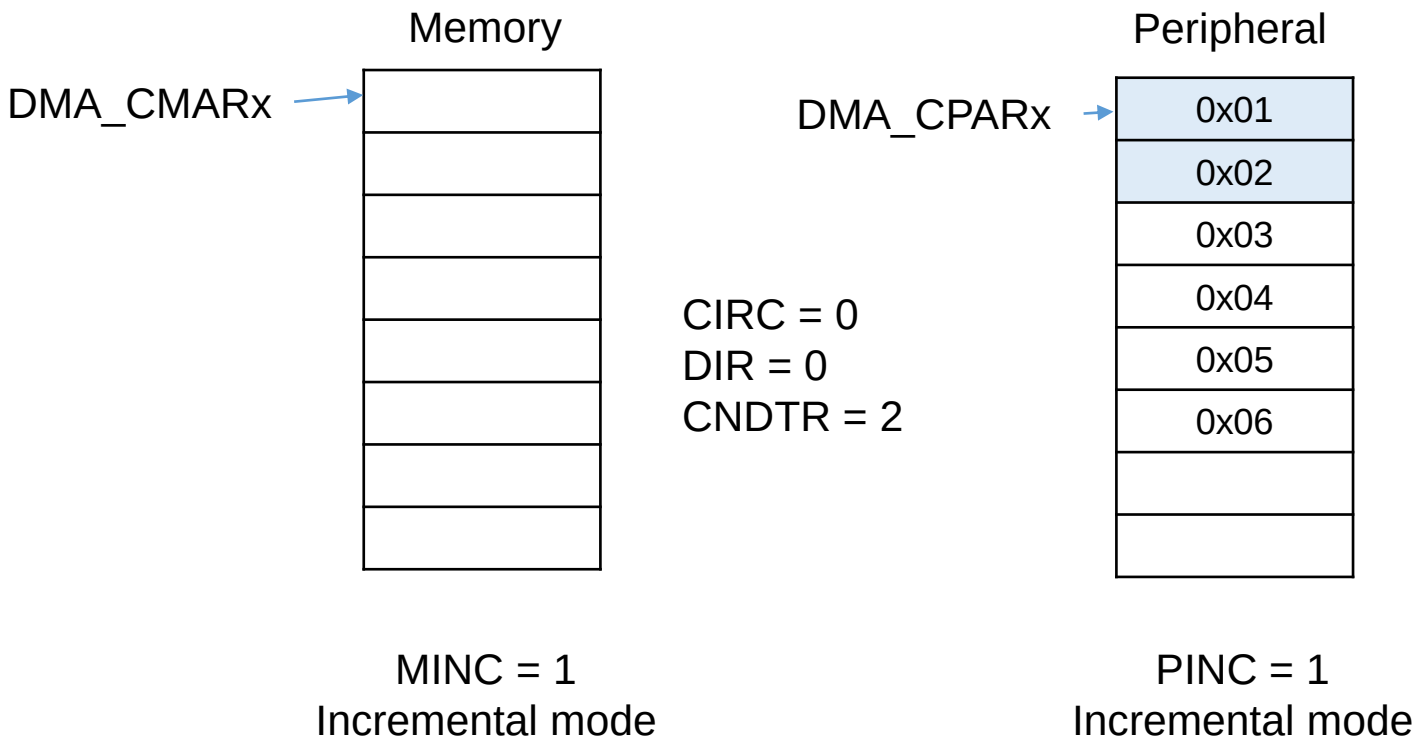
- Incremental mode in one side
  - MICN = 1, PINC = 0
  - MSIZE = 0: 8bit, PSIZE = 0: 8bit
- DMA stops since CNDTR is zero now.



baiyh@sustech.edu.cn

STM32 DMA Transfer

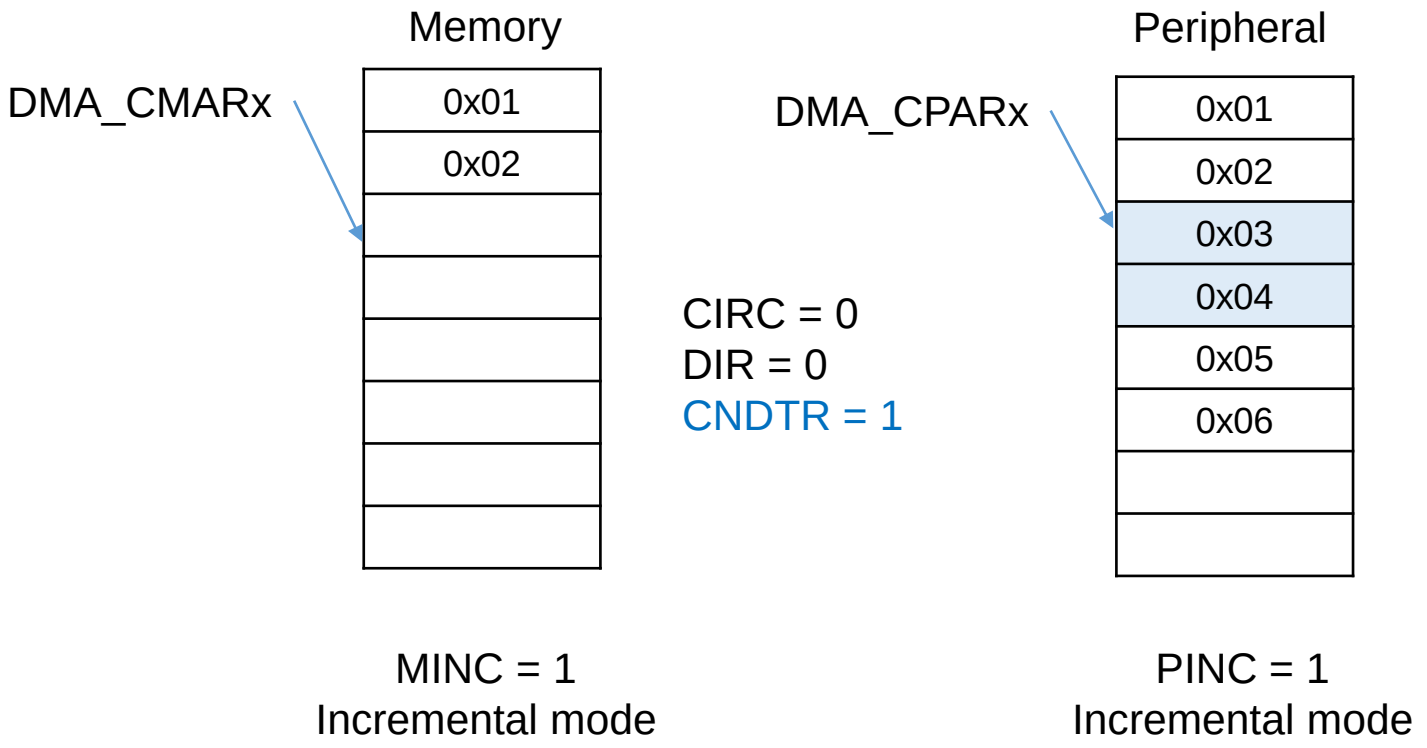
- Incremental mode in both sides
  - MICN = 1, PINC = 1
  - MSIZE = 1: 16bit, PSIZE = 1: 16bit



baiyh@sustech.edu.cn

STM32 DMA Transfer

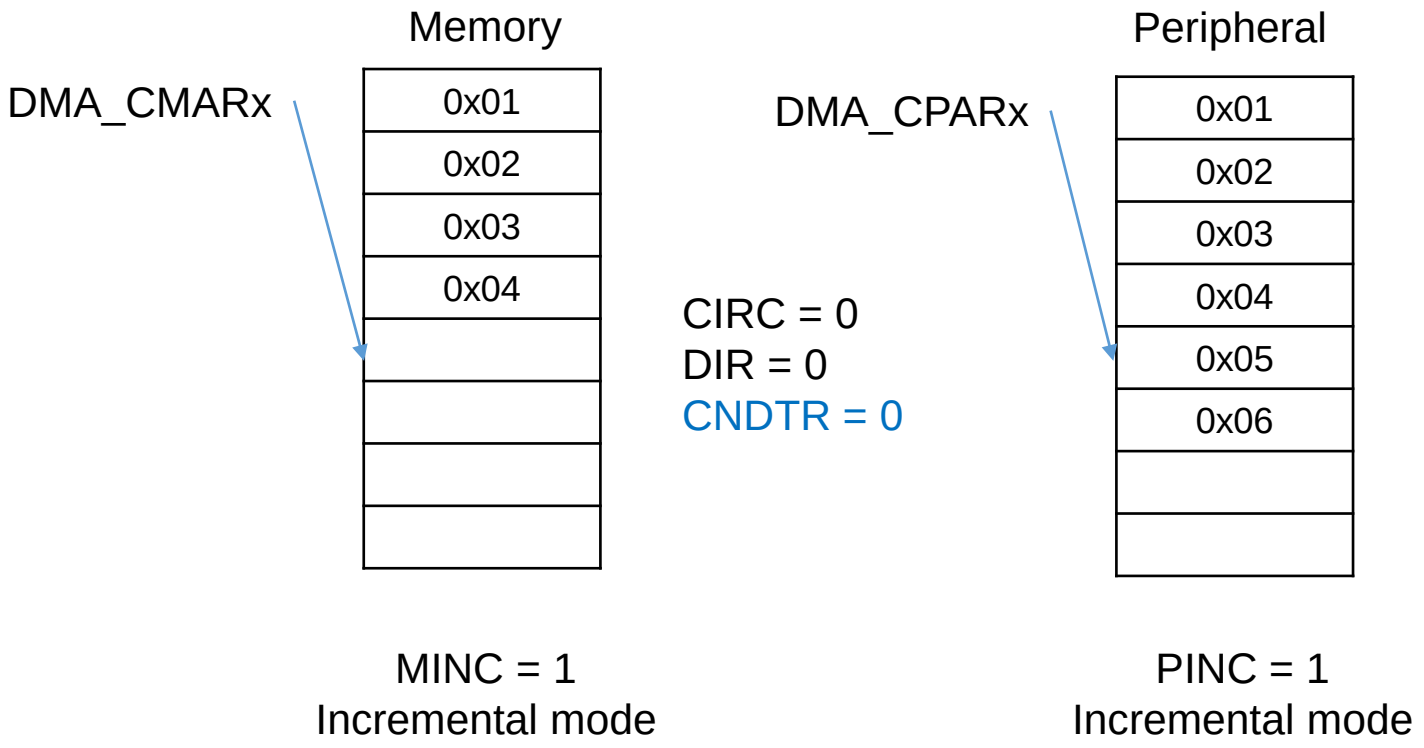
- Incremental mode in both sides
  - MICN = 1, PINC = 1
  - MSIZE = 1: 16bit, PSIZE = 1: 16bit



baiyh@sustech.edu.cn

STM32 DMA Transfer

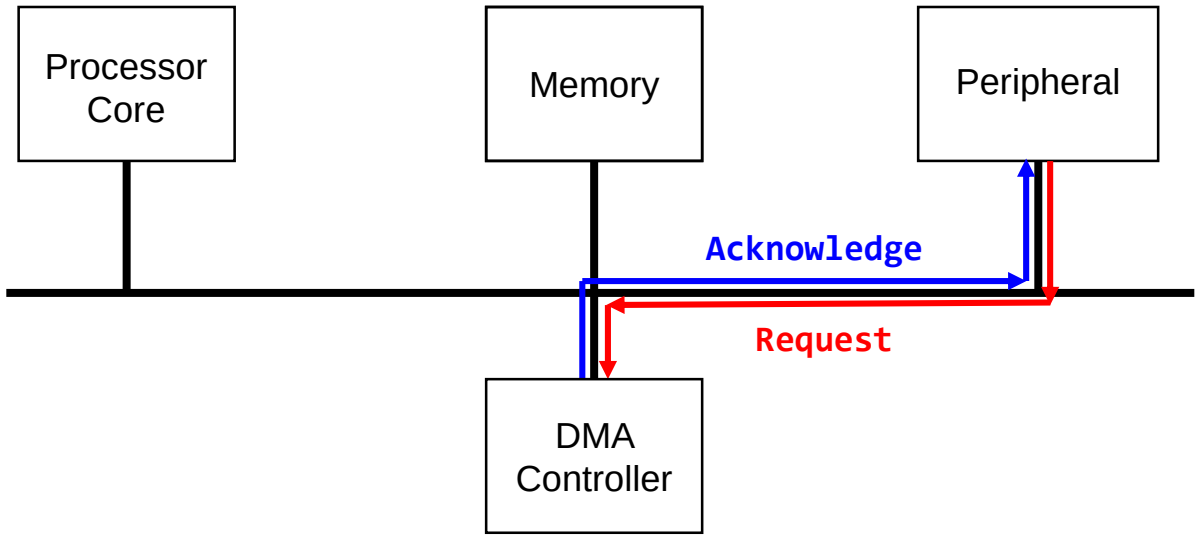
- Incremental mode in both sides
  - MICN = 1, PINC = 1
  - MSIZE = 1: 16bit, PSIZE = 1: 16bit
- DMA stops since CNDTR is zero now



baiyh@sustech.edu.cn

DMA Request

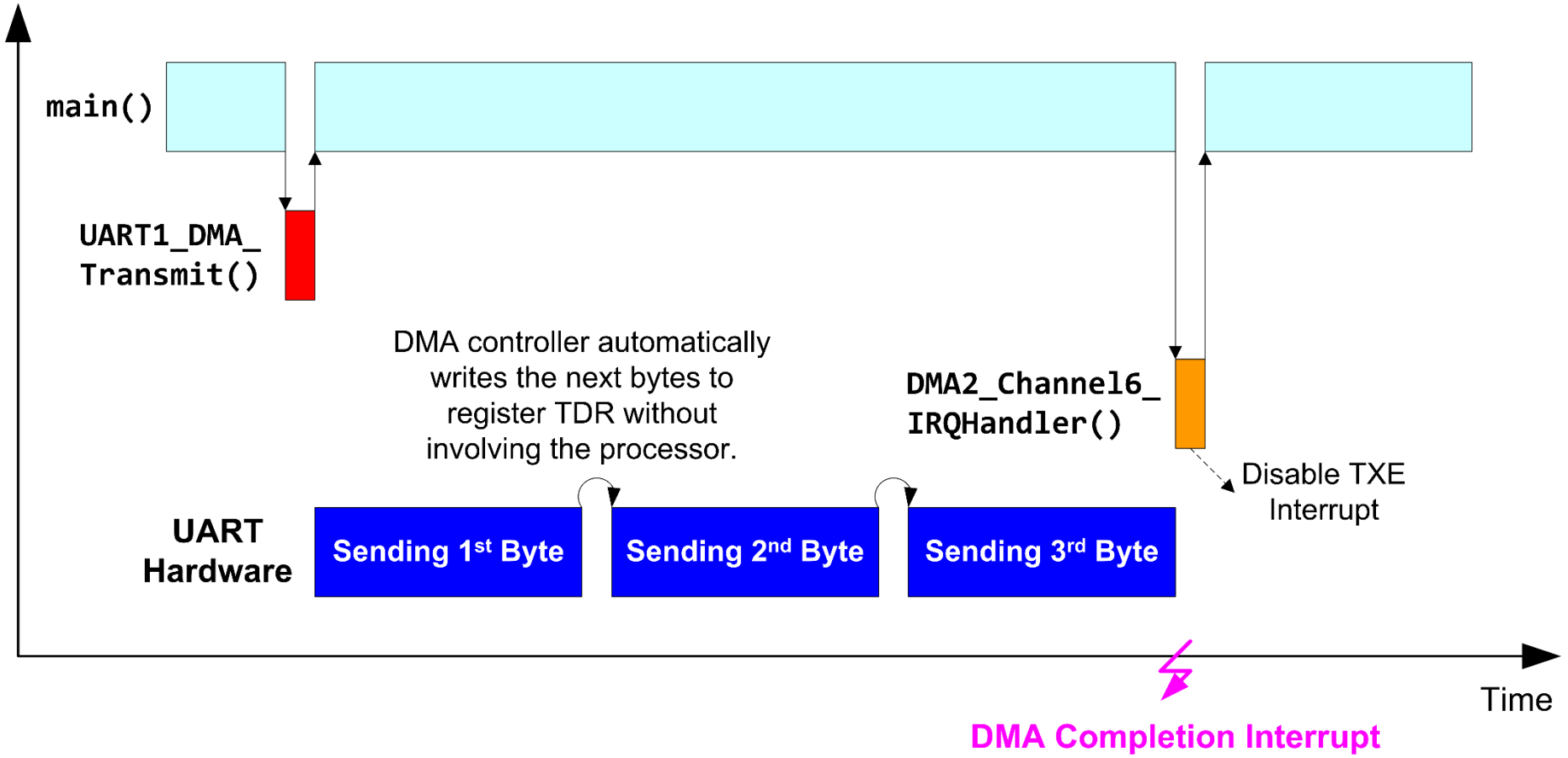
- When does the DMA transfer start?
  - When the peripheral is ready to send or receive data, the peripheral will generate a DMA request signal to the DMA controller to request a data transfer.



DMA Interrupts

- Programmable and Independent source and destination transfer data size:
  - Byte, Halfword or Word
- Three event flags:
  - DMA Half Transfer
  - DMA Transfer complete
  - DMA Transfer Error

UART DMA: Receiving & Sending



DMA Summary

- Without DMA, CPU must execute many load and store instructions, leading to slower performance.
- DMA, which makes an automatic data transfer when received a DMA request without involving CPU, accelerates the overall performance.

I/O interrupt transfer	**DMA Transfer**
Software control, needs context switch	Hardware control, no need of context switch
Data transfer speed is low	Data transfer speed is fast
CPU is involved in the transfer	CPU is not involved in the transfer
No need of extra hardware	DMA controller is required
Can used for exception handling	Only used for data transfer
Used for small data transfer	Used for large data transfer

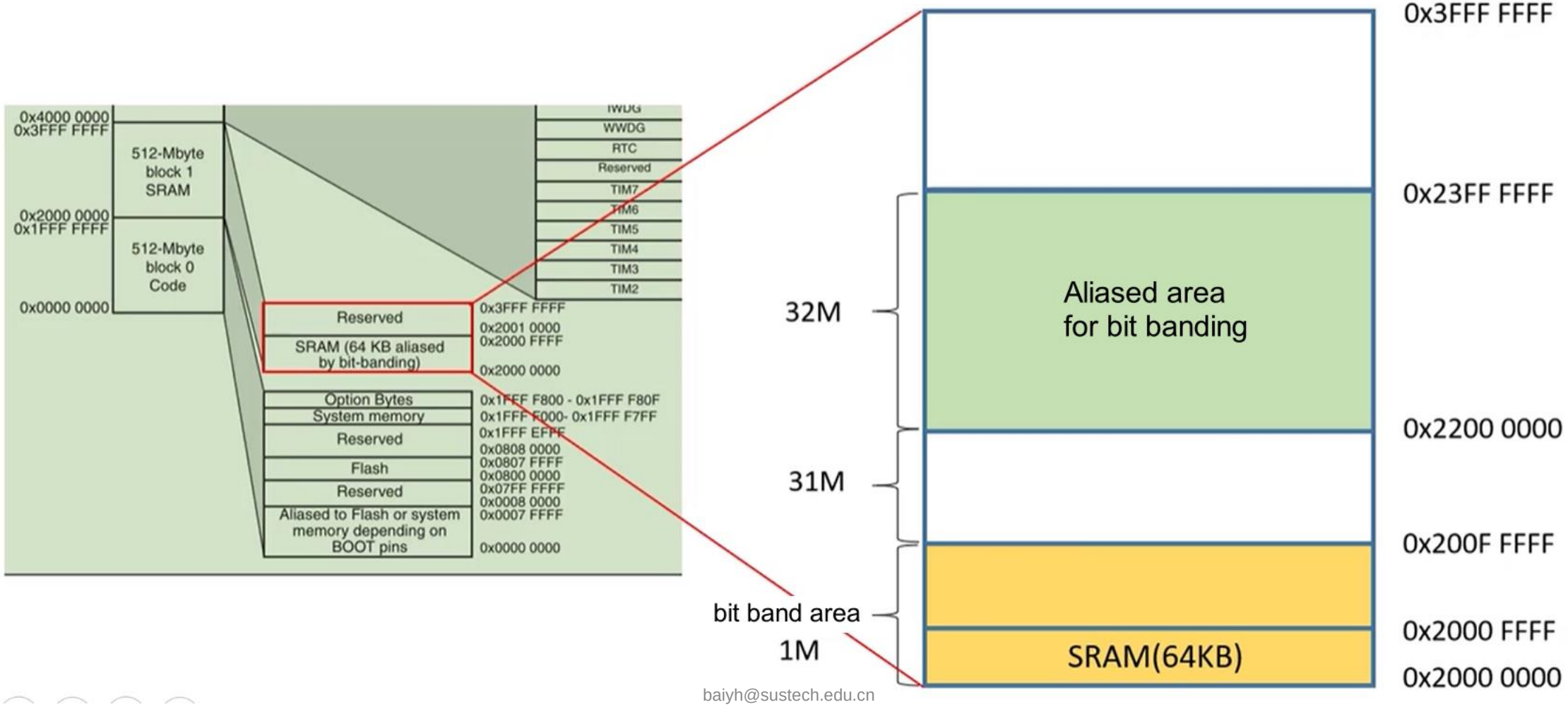
Outline

- DMA
- **Bit Banding**
- ARM Architecture - Pipeline

Bit Banding

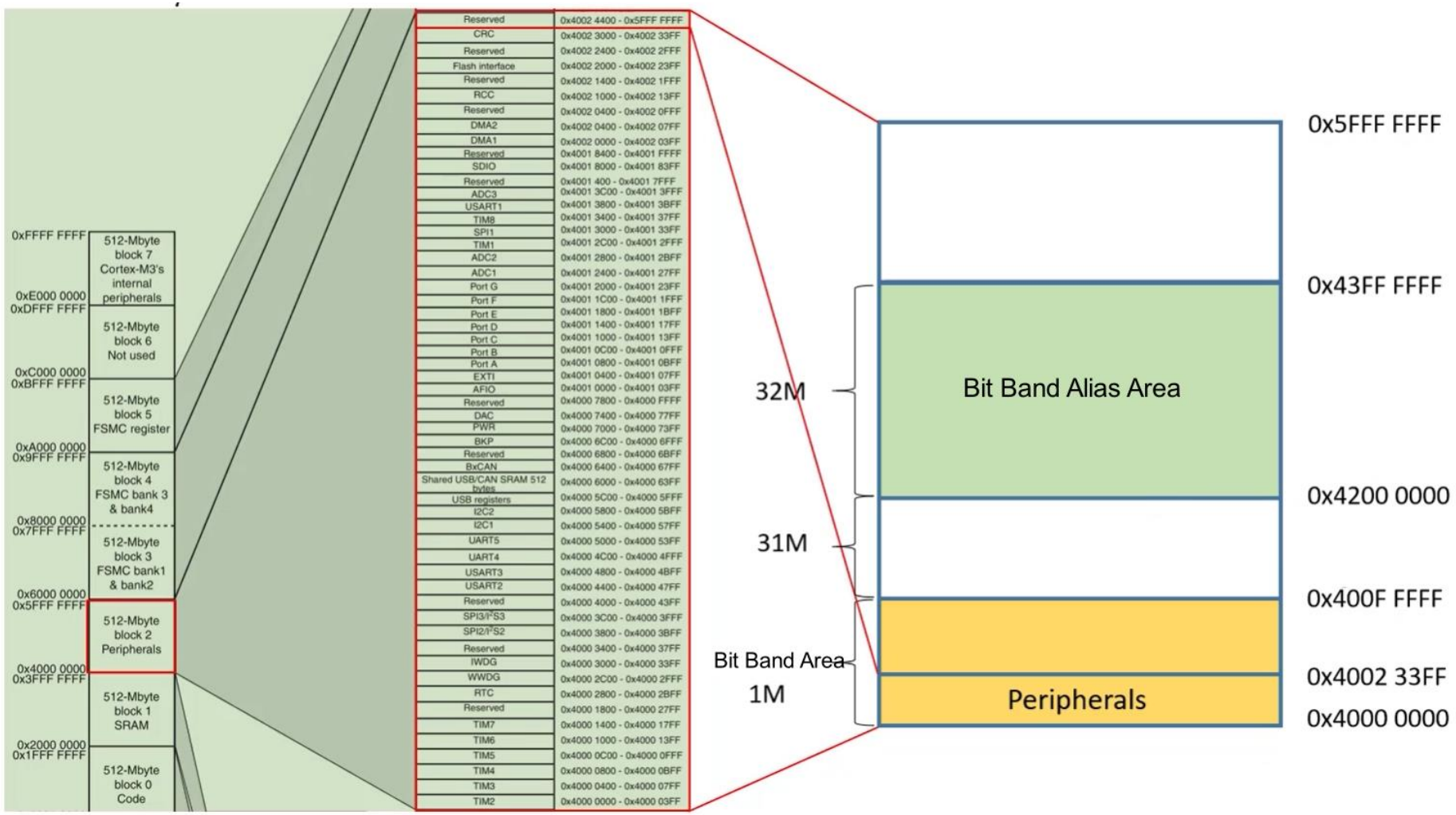
- The Cortex-M3 memory map includes two bit-band regions. These regions map each word in an alias region of memory to a bit in a bit-band region of memory. Writing to a word in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.
- In the STM32 both peripheral registers and SRAM are mapped in a bit-band region. This allows single bit-band write and read operations to be performed. The operations are only available for Cortex-M3 accesses, not from other bus masters (e.g. DMA).

Bit Banding for SRAM



baiyh@sustech.edu.cn

Bit Banding for Peripheral



Bit Banding Operation

- A mapping formula shows how to reference each word in the alias region to a corresponding bit in the bit-band region. The mapping formula is:
  - bit_word_addr = bit_band_base + (byte_offset x 32) + (bit_number × 4)
- where:
  - bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.
  - bit_band_base is the starting address of the alias region byte_offset is the number of the byte in the bit-band region that contains the targeted bit
  - bit_number is the bit position (0-7) of the targeted bit.

Bit Banding Operation

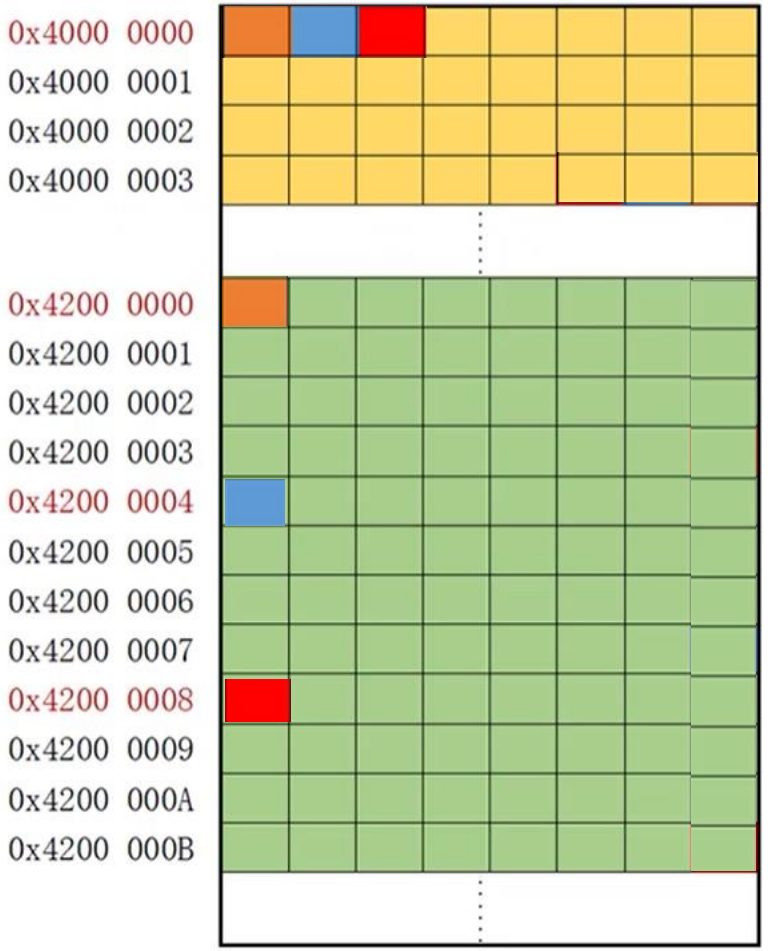
- Example:
  - The following example shows how to map bit 2 of the byte located at SRAM address 0x20000300 in the alias region:

    0x22006008 = 0x22000000 + (0x300*32) + (2*4).

  - Writing to address 0x22006008 has the same effect as a read-modify-write operation on bit 2 of the byte at SRAM address 0x20000300.

  - Reading address 0x22006008 returns the value (0x01 or 0x00) of bit 2 of the byte at SRAM address 0x20000300 (0x01: bit set; 0x00: bit reset).

Bit Banding Operation

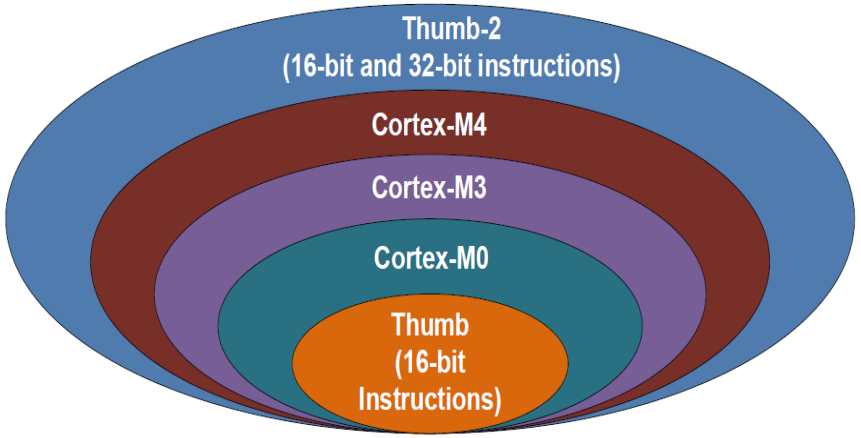


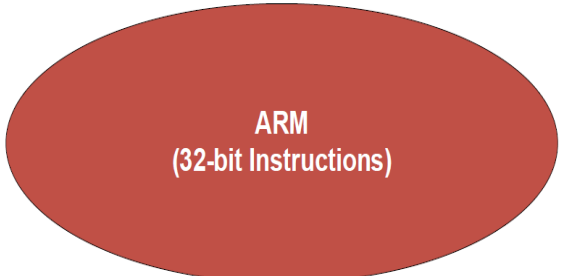
Outline

- DMA
- Bit Banding
- **ARM Architecture - Pipeline**

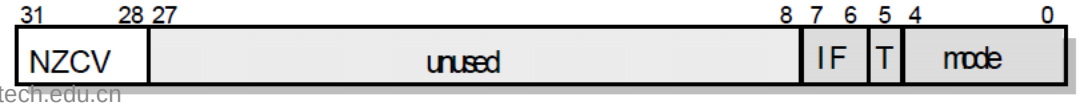
Instruction Sets

- Thumb instructions are 16 bit compressed ARM instructions.
  - ARM code is 40% faster than Thumb code if instructions are fetched on a 32 bit bus, so in a system where performance is paramount, ARM code and a 32 bit memory system are used.
  - However in a 16 bit memory system, Thumb code is 45% faster than ARM code. In a system where memory cost and power consumption are important then a 16 bit memory system and Thumb code would be the better choice.
  - Most systems use a bit of each; ARM code for the critical routines and Thumb code for background tasks.



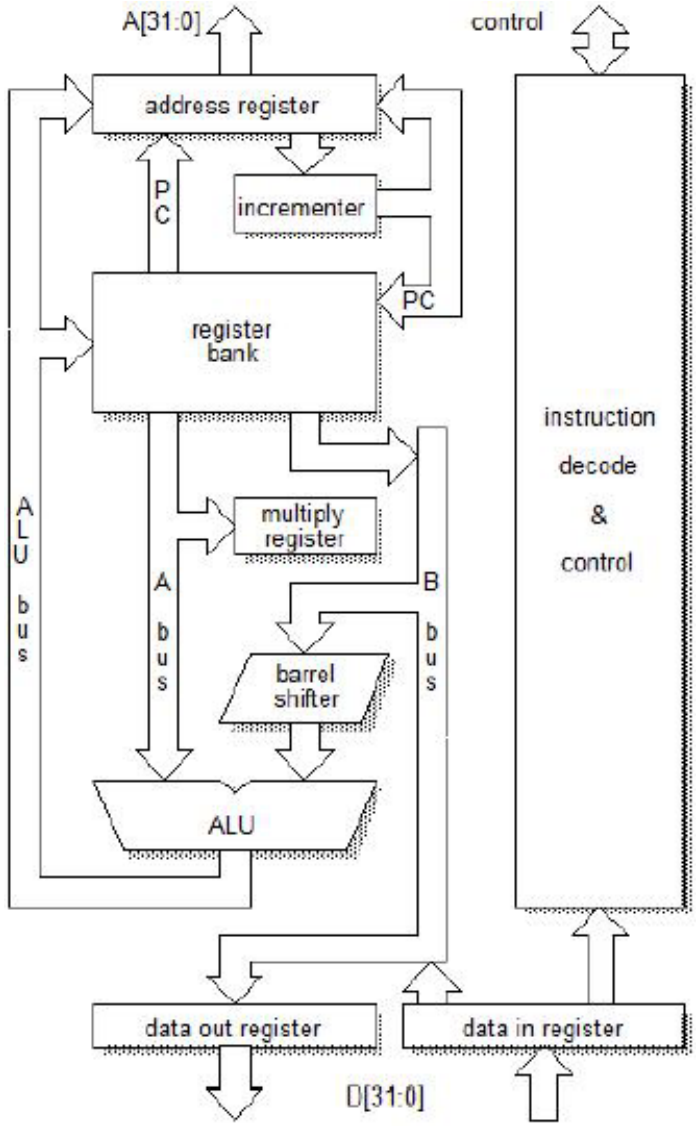


T: The thumb bit



Instruction Pipelines

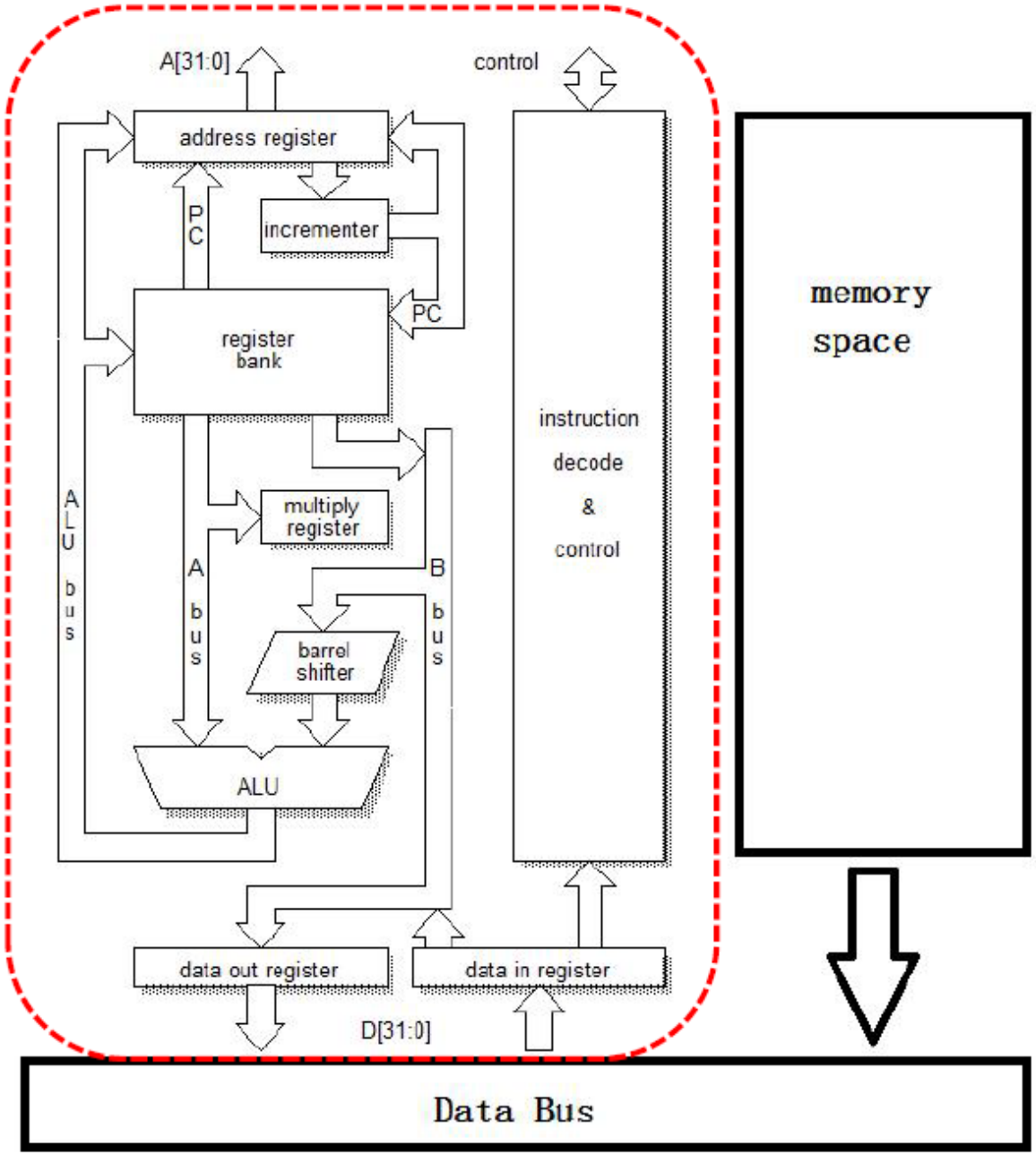
- Instruction pipelines is an important feature of all modern microprocessors.
- For the same basic speed of transistor operation, an n stage
- instruction pipeline allows the microprocessor to execute up to n times as many instructions in a given time.
- The ARM7 microprocessor has a three stage pipeline
  - one stage for each of the CPU cycles;
  - fetch, decode and execute.



baiyh@sustech.edu.cn

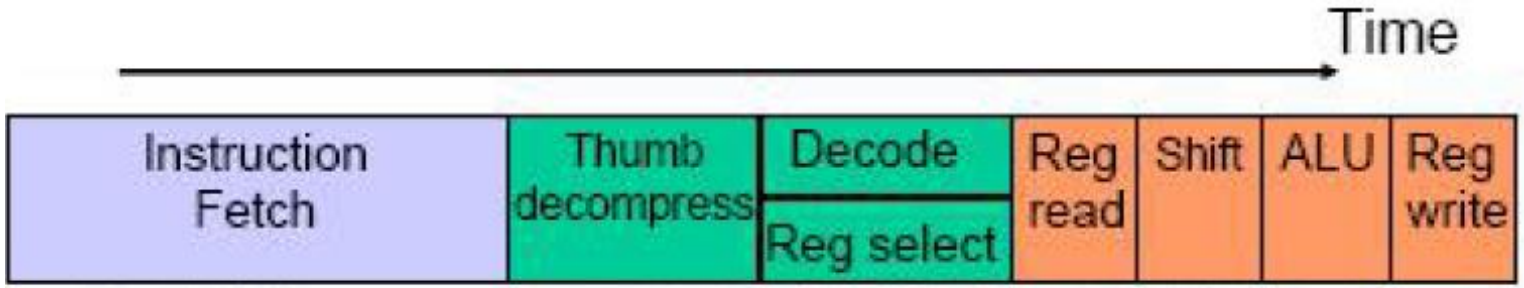
ARM7 3-stage pipeline

- Fetch
  - Increment PC
  - Read next instruction
- Decode
  - Control signal generation
  - Read from register file
- Execute
  - Arithmetic/logic operation
  - Calculate branch address
  - Load/store



ARM7 3 stage pipeline: detail

- In each stage of the ARM7 pipeline, several things happen; normally consecutively:



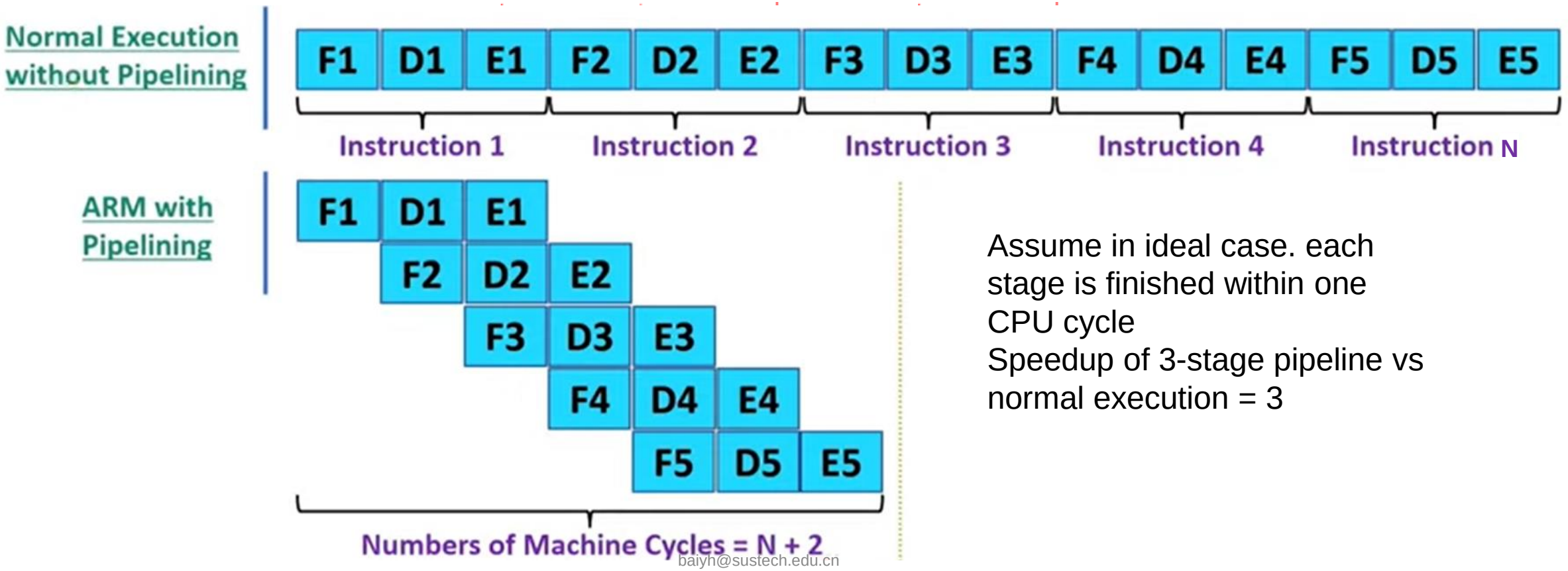
FETCH
Instruction fetched from memory.

DECODE
Thumb to ARM decompression.
Decode.
Register selection

EXECUTE
Register read.
Barrel shift.
ALU operation.
Register write.

ARM7 3 stage pipeline

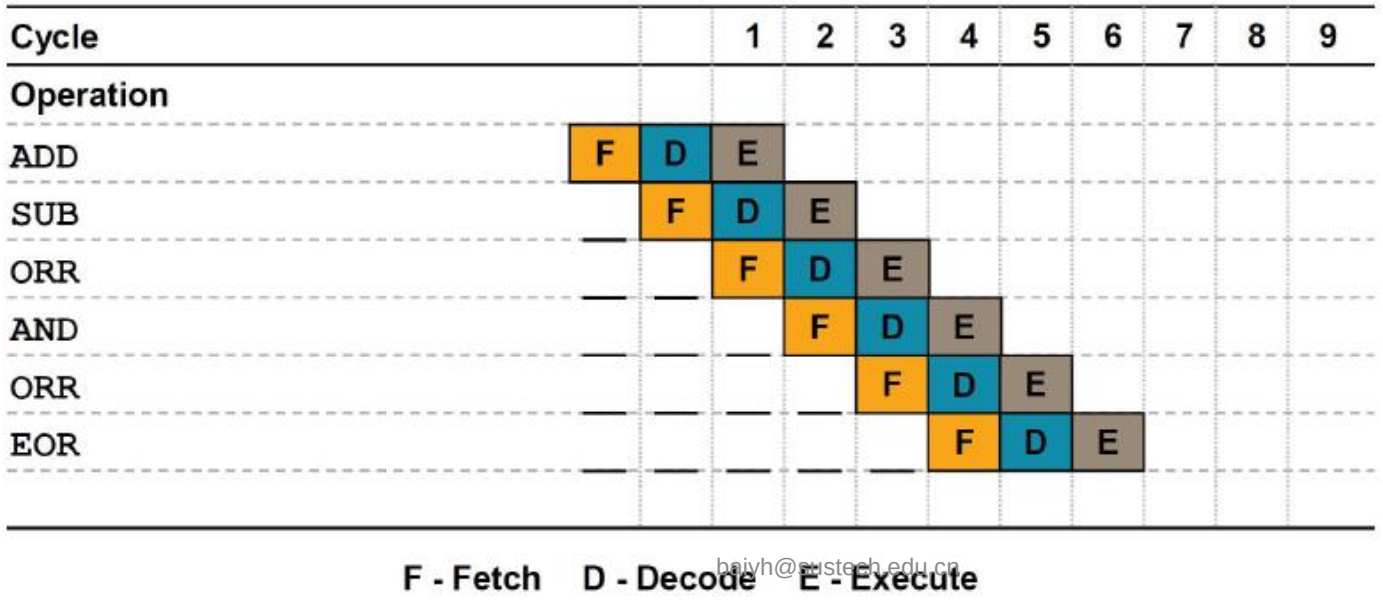
- In a three stage pipeline, the CPU can simultaneously execute an instruction, decode the next instruction and fetch the next instruction.



Assume in ideal case. each stage is finished within one CPU cycle
Speedup of 3-stage pipeline vs normal execution = 3

Pipelines with optimum operation

- A pipeline operates optimally if the instructions to be executed are in consecutive memory locations and no conflict occurs on the data bus.
- When this is the case, the microprocessors operates at one clock cycle per instruction (1 CPI).
- The best performance cannot be achieved if a branch or load instruction is executed or if an interrupt is serviced.



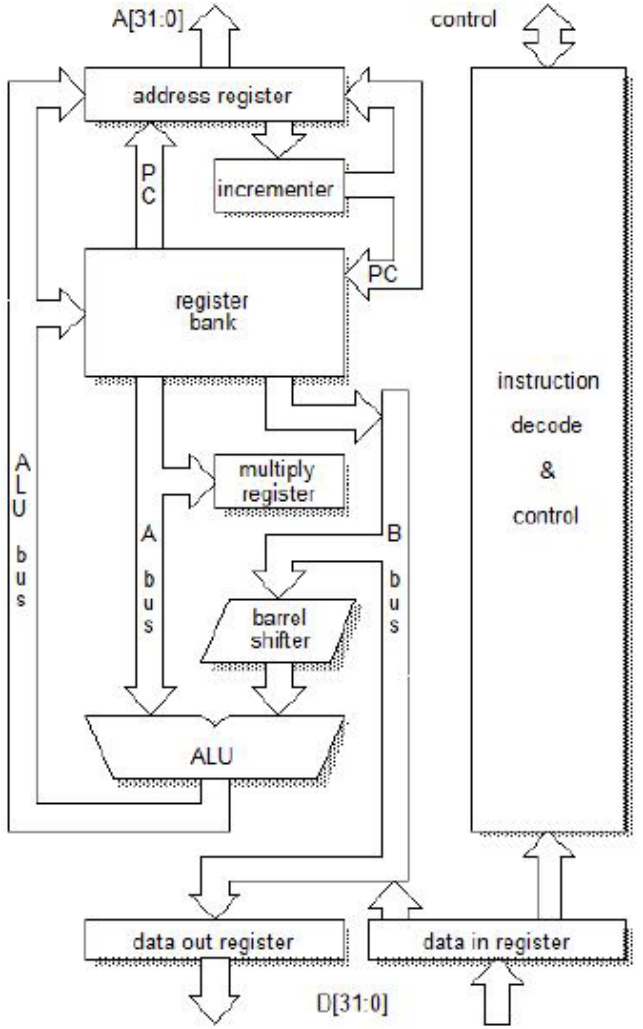
Pipelines with Load and store instructions

- Load and store instructions use the data bus to pass data to or from memory so that the data bus cannot be used to fetch an instruction at the same time.

Cycle	1	2	3	4	5	6	7	8	9
Operation									
ADD	F	D	E						
SUB		F	D	E					
LDR			F	D	E	M	W		
AND				F	D	S	S	E	
ORR					F	S	S	D	E
EOR							F	D	E

F - Fetch D - Decode E – Execute M – Memory W – Writeback S - Stall

Structural hazards: different instructions in different stages (or thesame stage) conflicting for the same resource



baiyh@sustech.edu.cn

Load and store instructions

- Data and instruction memory cannot be accessed simultaneously due to the bus conflict. While LDR is in the memory access cycle, the next instruction’s fetch is blocked.
- So in average, 8 clock cycles are used to execute 6 instructions. CPI = 1.3 clock cycles.(count from the execute stage of the first instruction to the execute stage of the last instruction).

Pipelines with Branch Instruction

- A branch instruction will reload the program counter so that two cycles are lost
  - e.g. assume that the instruction at address 0x8000 is Branch to 0x8FEC

Cycle				1	2	3	4	5	6	7	8	9
Address	**Operation**											
0x8000	BL 0x8FEC	F	D	E	L	A						
0x8004	SUB		F	D								
0x8008	ORR			F								
0x8FEC	AND				F	D	E					
0x8FF0	ORR					F	D	E				
0x8FF4	EOR						F	D	E			

These instructions must be fetched in case it was a conditional branch

F - Fetch D - Decode E – Execute L – Linkret A - Adjust

Control hazard: fetch cannot continue because it dows not know the outcome of an earlier branch

Branch Instruction

- So in average, in six clock cycles only four instructions are executed – 1.5 CPI (count from the execute stage of the first instruction to the execute stage of the last instruction).
- For a branch and link instruction the link register is updated during the two clock cycles when no instruction is being executed.
- If a conditional branch is not executed then no clock cycles are lost.

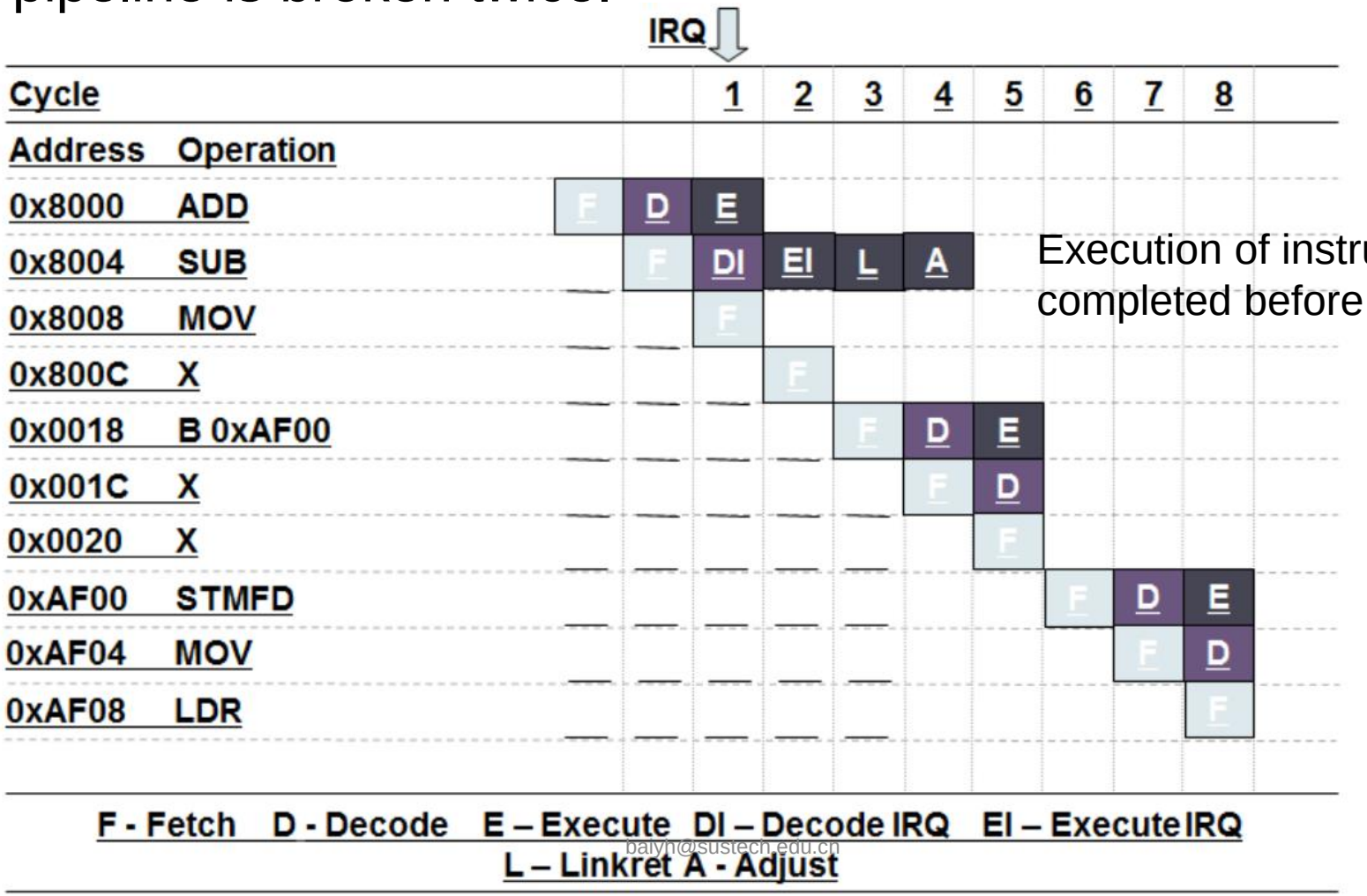
Question

- How many clock cycles do the following instructions take to execute if the value in r0 is
  - a) not equal to 0 and
  - b) equal to 0?

```
     movs r0, r0
     bne cont
    add r3, r4, r5
    eor r2, r1, r4
    sub r9, r8, r7
cont: sub r6, r7, #42
```

Pipelines with Interrupts

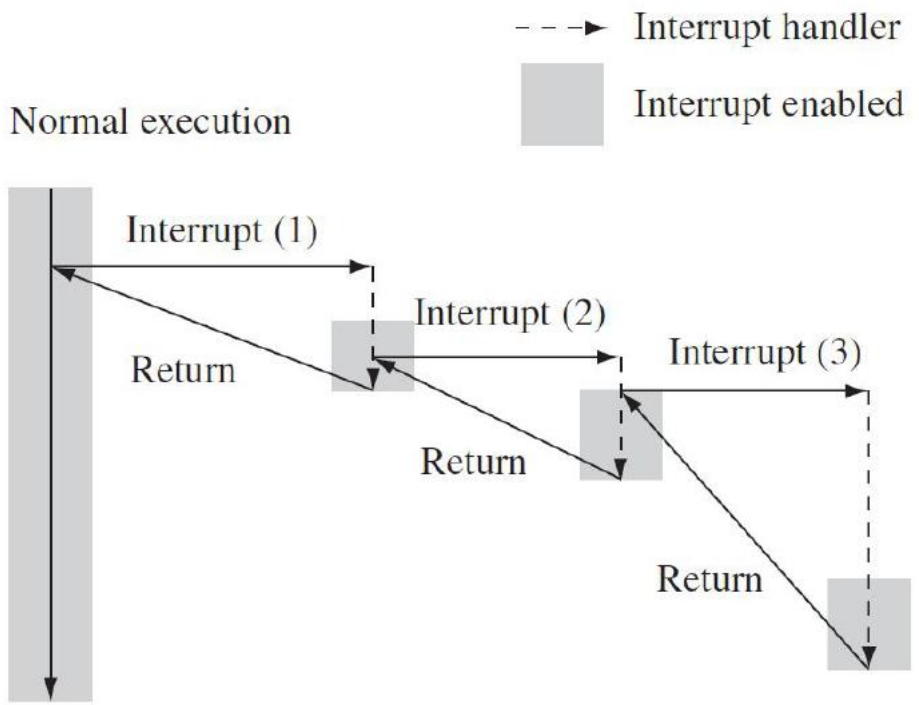
- An IRQ reloads the program counter with 0x0018 and then branches so that the pipeline is broken twice.



Execution of instruction at 0x8000 is completed before IRQ is serviced

Interrupt latency

- The latency is the time between the microprocessor receiving an interrupt signal and the first instruction being executed (i.e. the instruction at 0xAF00 in example).
- The minimum latency for an IRQ is 7 clock cycles (count from the fetch stage of the first instruction to the fetch stage of the 0xAF00 instruction)
  - minimum because an IRQ could be interrupted by other IRQs (if nested interrupts allowed) and an FIQ.



A three-level nested interrupt.

Eliminating bus conflicts

- Bus conflicts could be eliminated by using two separate data buses; one for instructions and one for load and store data - this is called a Harvard architecture.
- A microprocessor, such as the ARM7, that uses one data bus for both instructions and data is said to have the von Neumann architecture.