ARM General Overview

ARMSchool Seminars 2005

2005-10-25

Powering the Supply Chain.℠

ARM – The company

- Develops the ARM range of RISC processor cores
- Licenses its RISC microprocessor core and SoC IP to a network of partners; semiconductor and system companies
- ARM does not manufacture silicon itself
- Also licenses architectural extensions, development tools, peripheral IP and SoC solutions
- ARM's market share of the embedded RISC microprocessor market is approx. 75% and to date, ARM Partners have shipped more than one billion ARM core-based microprocessors

ARM – Architecture

- v4T
  Introducerade Thumb – vanligast idag.
- v5TE
  1999 – DSP instruktioner.
- v5TEJ
  2000 – ”Jazelle” Java VM co processor
- v6
  2001 – SIMD media extensions

ARM – Families

- ARM7 Thumb (v4T)
  von Neuman, 3 steg pipe,
  - ARM7TDMI (-S)
  - ARM720T (MMU, 8K Cache)
  - ARM7EJ-S (DSP, Jazelle)
- ARM9 Thumb (v4T)
  Harvard, 5 steg pipe, Cache, MMU
  - 920T (Dual 16K Cache)
  - 922T (Dual 8K Cache)
  - 940T (Dual 4K Cache, MPU)
- ARM9E (v5TEJ)
  - ARM926EJ-S (upto 128K Cahe, MMU, Dual AHB)
  - ARM946E-S (upto 1M Cache, MPU)
  - ARM966E-S (No Cache, no MMU)
- ARM10E (v5TEJ)
- ARM11 (v6)

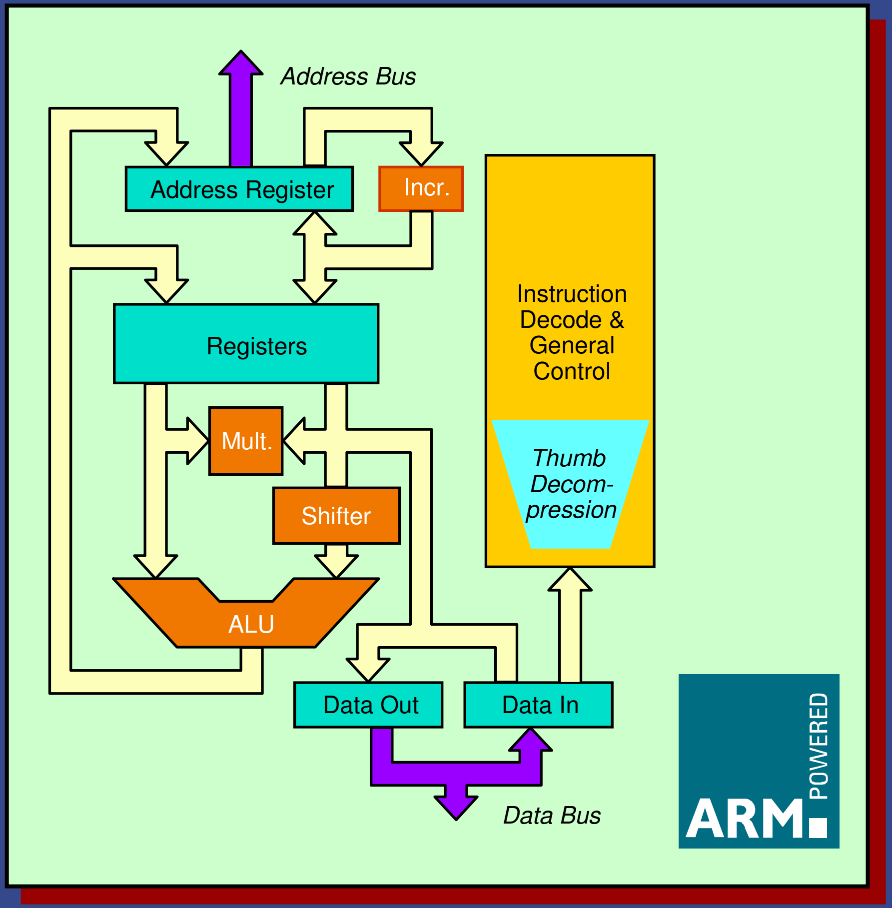
ARM7TDMI-S

The ARM7TDMI-S is based on ARM7 core

- 3 stage pipeline
- Von Neumann architecture
- CPI ~1.9
- **T:** Thumb instruction set
- **D:** includes debug extensions
- **M:** enhanced multiplier (32x8) with instructions for 64-bit results
- **I:** core has EmbeddedICE logic extensions
- **S**: fully synthesisable (soft IP)

Bus Width

- ARM is a 32-bit architecture
  - Data pathes and (**ARM**) instructions are 32 bits wide
  - Von Neumann architecture
    - instructions and data use the same 32-bit data bus
  - There is a subset of 16-bit instructions (**Thumb**) optimized for code density*

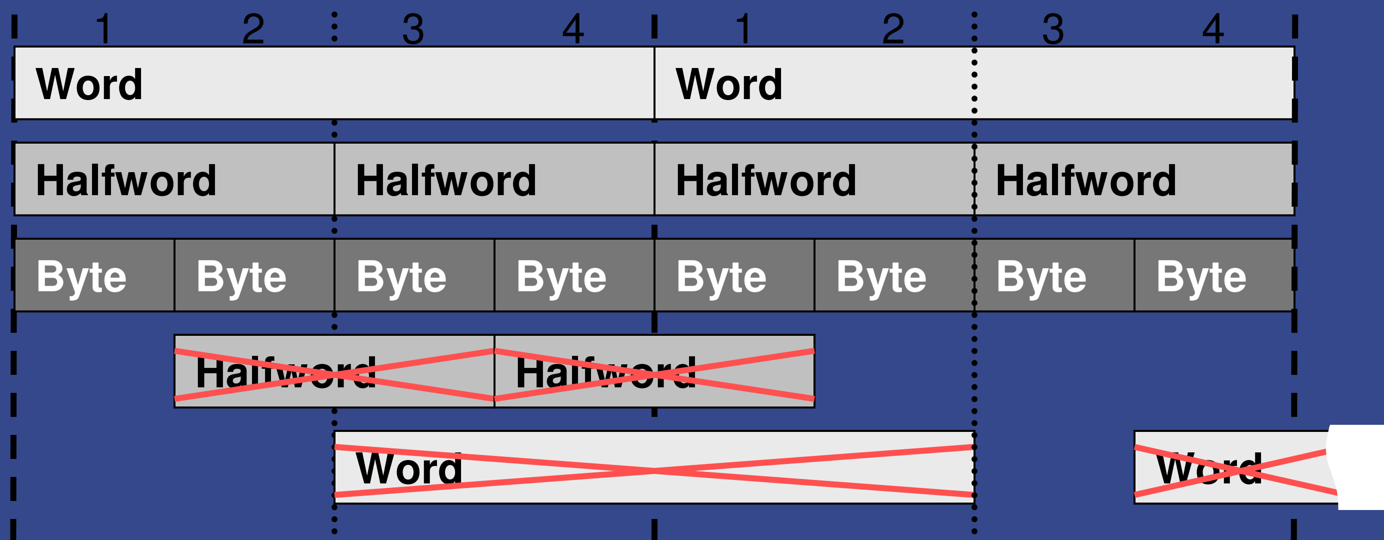


Thumb State

- Set of instructions re-coded into 16 bits
  - Improved code density by ~ 30%
  - Saving program memory space
  - Higher performance (up to 60%) when running from 16-bit wide external memory
- In Thumb state only the program code is 16-bit wide
  - After fetching the 16-bit instructions from memory, they are de-compressed to 32 bit instructions before they are decoded and executed
  - All **operations are still 32-bit** operations

Data Types and Alignment

- Definitions:
  - Word = 32 bits (four bytes)
  - Halfword = 16 bits (two bytes)
  - Byte = 8 bits



Processor Modes

ARM has seven operating modes

- **User** unprivileged mode under which most applications run
- **FIQ** entered, when a high priority (fast) interrupt is raised
- **IRQ** general purpose interrupt handling
- **Supervisor** protected mode for the operating system
  entered on reset or software interrupt instruction
- **System** privileged mode using the same registers as user mode
  (not in ARM architectures 1, 2 and 3)
- **Abort** used to handle memory access violations
- **Undefined** used to handle undefined instructions

Registers (1)

An ARM core has 37 registers (32-bits wide)

- General purpose registers
  - 1 program counter
  - 30 general purpose registers
- Status registers
  - 1 current program status register
  - 5 saved program status registers

These registers are not all accessible at the same time. The processor state and operating mode determine which registers are available to the programmer.

Registers (2)

- Depending on processor mode one of several banks is accessible. Each mode can access
  - the program counter **r15** (**PC**)
  - a particular **r13** (stack pointer **SP**)
  - a particular **r14** (subroutine link register, **LR**)
  - a particular set of **r0-r12** registers
  - the current program status register (**CPSR**)
- Privileged modes (except Sytem mode) can also access
  - a particular **SPSR** (saved program status register)

Register Banking

User and System

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (SP)
r14 (LR)
r15 (PC)
CPSR

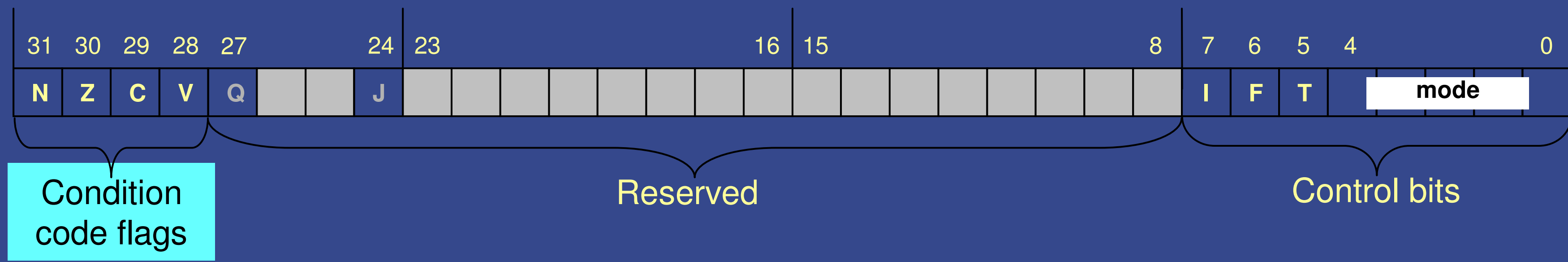
Banked registers

FIQ	IRQ	Supervisor	Abort	Undefined
r8_fiq				
r9_fiq				
r10_fiq				
r11_fiq				
r12_fiq				
r13_fiq (SP)	r13_irq (SP)	r13_svc (SP)	r13_abt (SP)	r13_und (SP)
r14_fiq (LR)	r14_irq (LR)	r14_svc (LR)	r14_abt (LR)	r14_und (LR)
SPSR_fiq	SPSR_irq	SPSR_svc	SPSR_abt	SPSR_und

Registers in Thumb State

- The Thumb state register set is a subset of the ARM state set. The programmer has direct access to:
  - eight general registers r0 - r7
  - the program counter PC
  - a Stack pointer SP
  - a Link register LR
  - the current program status register CPSR
- In Thumb state, the high registers (r8 - r15) are not part of the standard register set. The assembly language programmer has limited access to them, but can use them for fast temporary storage

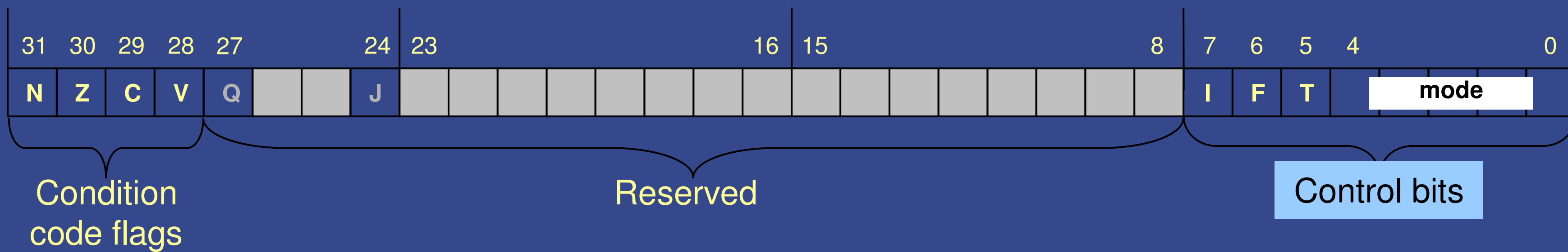
Program Status Register (1)



- Condition Code Flags
  - N: Negative or less than
  - Z: Zero
  - C: Carry or borrow or extend
  - V: Overflow

To not disturb reserved bits, a read-modify-write strategy should be applied to change PSR bits.

Program Status Register (2)



- Interrupt Disable Bits
  - I: IRQ interrupts **disable**
  - F: FIQ interrupts **disable**
- T Bit
  - Thumb mode (when set)
  - ARM mode (when cleared)

- Mode Bits

10000 (0x010)	User
10001 (0x11)	FIQ
10010 (0x12)	IRQ
10011 (0x13)	Supervisor
10111 (0x17)	Abort
11011 (0x1B)	Undefined
11111 (0x1F)	System

Program Counter (r15)

- When the processor is executing in **ARM** state
  - all instructions are 32 bits wide
  - all instructions must be word aligned
  - bits [31:2] contain the PC, bits [1:0] are zero

    (instructions cannot be halfword or byte aligned)
- When the processor is executing in **Thumb** state
  - all instructions are 16 bits wide
  - all instructions must be halfword aligned
  - bits [31:1] contain the PC, bit [0] is zero

    (instructions cannot be byte aligned)

Exception Handling

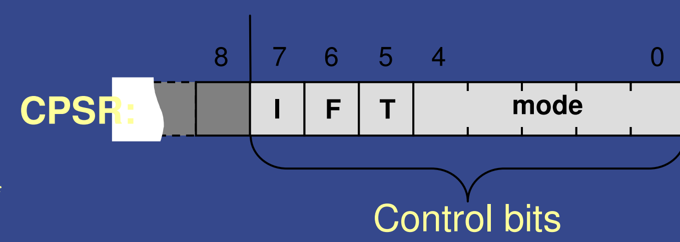
- Entering an exception the ARM core
  - saves the address of the next instruction in the appropriate LR

r15 (PC) —PC + 4 or PC + 8→ r14_<mode> (LR)

  - copies the CPSR into the appropriate SPSR

CPSR → SPSR_<mode>

  - sets appropriate CPSR bits
    - interrupt disable bits
    - mode field bits
    - if running in Thumb state, enter ARM state*



  - forces PC to fetch next instruction from relevant exception vector

*: all exceptions are handled in ARM state!

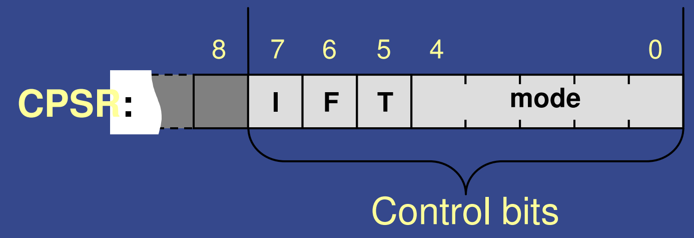
Exception Vectors

	⋮
0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Leaving Exception

- To leave an exception, the exception handler must
  - copy SPSR back into CPSR

(automatically restoring also I, F and T)



  - move contents of current LR minus offset* to PC

*: varies according to type of exception: 2, 4 or 8

Instruction Set

- All instructions are 32-bits long
- Many instructions execute in a single cycle
- Instructions are **conditionally** executed
- ARM is a load / store architecture
  – via registers => RISC
- Load or store multiple registers in a single instruction

  using <register list>

Instruction Examples

- Data processing instructions

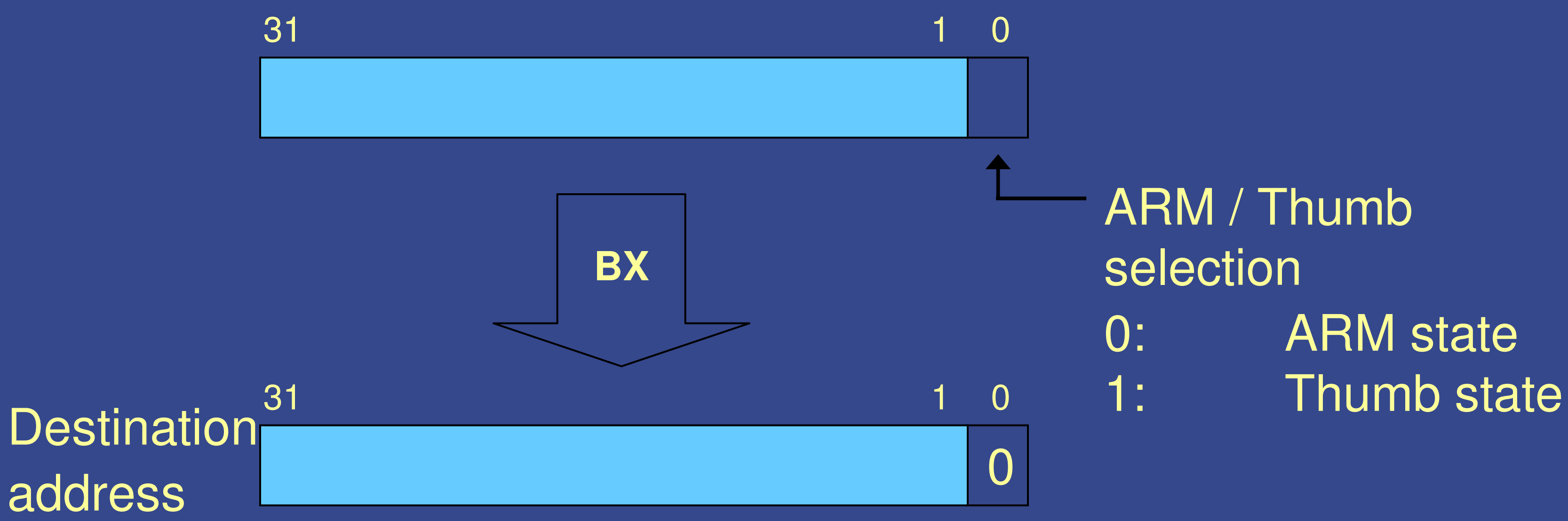
– SUB	r0, r1, #5	*r0 := R1 - 5*
– ADD	r2, r3, r3, LSL #2	*r2 := r3 + (r3, LSL #2)*
– ADDS	r4, r4, #0x20	*r4 := r4 + 32 and set flags*
– ADDEQ	r5, r5, r6	*r5 := r5 + r6 if equal*

- Specific memory access instructions

– LDR	r0, [r1, #4]	*r0 := [r1 +4]*
– STRNEB	r2, [r3, r4]	*[r3 + r4] := r2 Byte operation if Z = 0; ignores r2[31:8]*
– LDRSH	r5, [r6, #2]!	*r5 := [r6 + 2] Halfword sign-ext. set bit [31:16] to bit 15 then r6 := r6 + 2*

ARM and Thumb Interworking

- Switch between ARM state and Thumb state using BX instruction
  - In ARM state: BX<condition> Rn
  - In Thumb state: BX Rn



3-Stage Instruction Pipeline

ARM	Thumb	Stage	Description
PC	PC	**Fetch**	Instruction Fetched from Memory
PC - 4	PC - 2	**Decode**	Thumb only: Thumb instruction decompressed to ARM instruction Instruction decoded
PC - 8	PC - 4	**Execute**	Registers read from Register Bank, Shift and ALU operations performed, Registers written back to Register Bank

3-Stage Endianness

- Little endian (Philips)

31	24 23	16 15	8 7	0

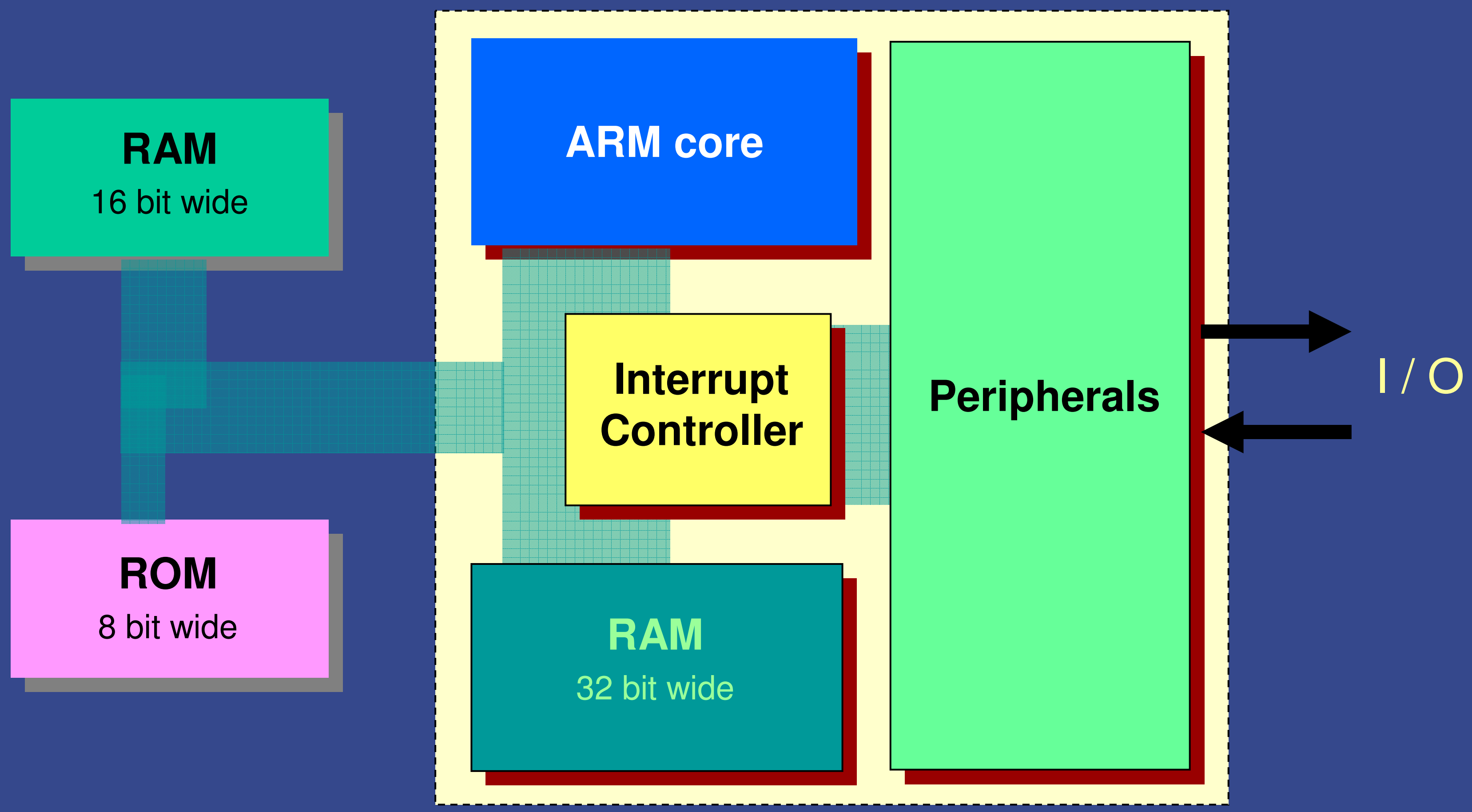
Word at address A			
Halfword @ address A+2		Halfword @ address A	
Byte @ address A+3	Byte @ address A+2	Byte @ address A+1	Byte @ address A

- Big endian

31	24 23	16 15	8 7	0

Word at address A			
Halfword @ address A		Halfword @ address A + 2	
Byte @ address A	Byte @ address A+1	Byte @ address A+2	Byte @ address A+3

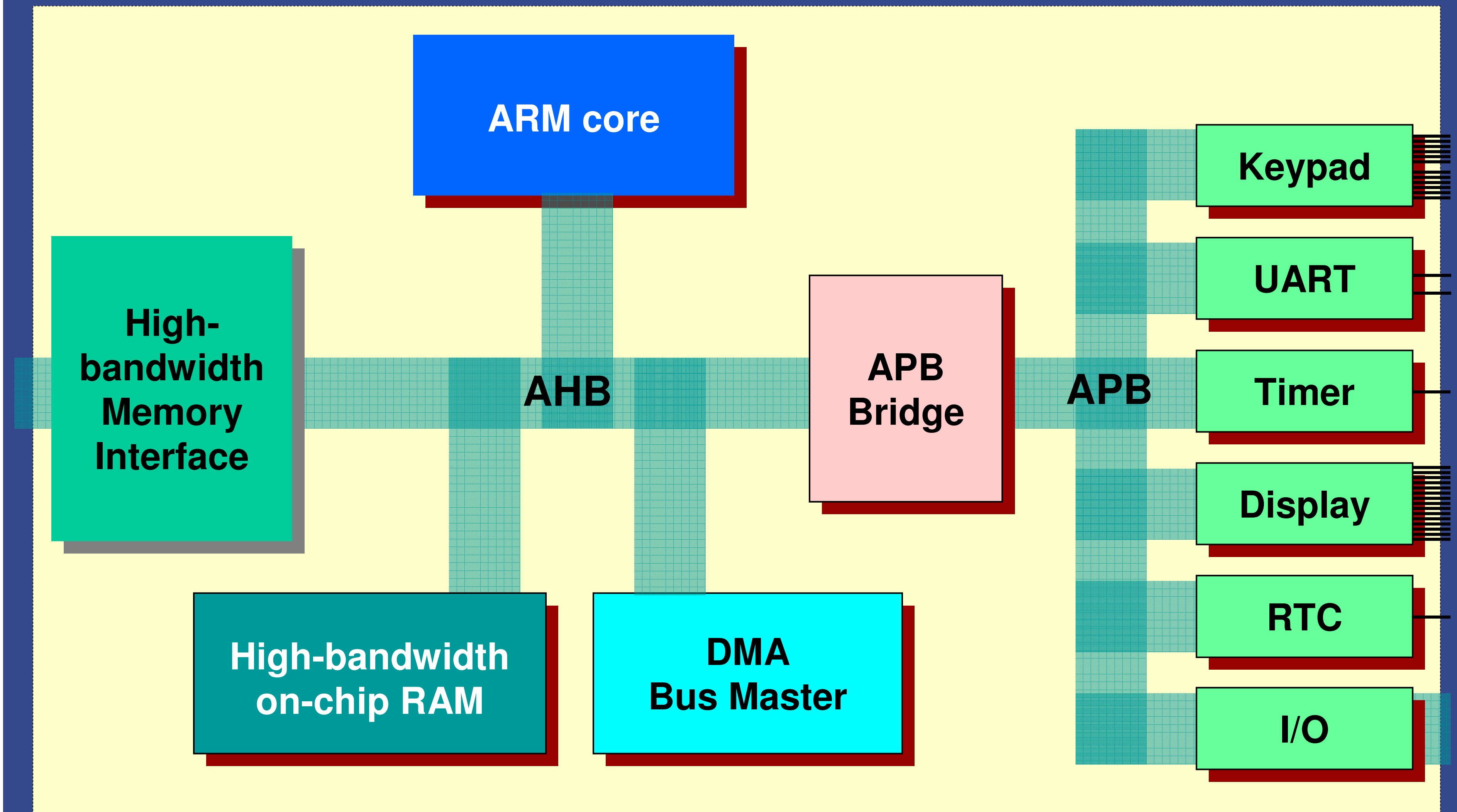
Example ARM based System



AMBA

- **A**dvanced **M**icrocontroller **B**us **A**rchitecture
  - on-chip interconnect
  - established, open specification
  - framework for SoC designs
  - enabler for IP reuse
  - ‘digital glue’ that binds IP cores together

Example AMBA System



AHB and APB / VPB

- **A**dvanced **H**igh-Performance **B**us
  - high-performance
  - pipelined
  - fully-synchronous backbone
  - multiple bus masters
- **A**dvanced **P**eripheral **B**us / **V**LSI **P**eripheral **B**us
  - low-power
  - non-pipelined
  - simple interface
  - wait support (VPB)

The End

Questions?