ARM® Processor
Programmer Model
aka
Instruction Set Architecture

An assembly language programmer’s view of the processor hardware

Instruction Set Architecture

• Programmer’s do care about:
  – What happens when a “branch” is executed?
  – What happens when a subroutine is called?
  – What happens when an interrupt occurs?

• Programmers Don’t Care so much about:
  – How is the processor control unit structured?
  – What is architecture of the cache memory?
  – What is the memory decoding architecture?
  – What is the I/O bus protocol?

• We care about BOTH since this “Microcontroller Architecture and Interfacing”
ISA Versions and Part Numbers

• Confusing since Acorn produced ARM1 and ARM2 then Advanced Risc Machines started with ARM6 Part numbers

• ISA Versions More or Less in Sequential order

• Be careful when referring to the “version” of ARM, make it clear if you mean part numbers or architectures

<table>
<thead>
<tr>
<th>ARM Family</th>
<th>ARM Architecture</th>
<th>ARM Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM1</td>
<td>ARMv1</td>
<td>ARM1</td>
</tr>
<tr>
<td>ARM2</td>
<td>ARMv2</td>
<td>ARM2</td>
</tr>
<tr>
<td>ARM2</td>
<td>ARMv2a</td>
<td>ARM250</td>
</tr>
<tr>
<td>ARM3</td>
<td>ARMv2a</td>
<td>ARM3</td>
</tr>
<tr>
<td>ARM6</td>
<td>ARMv3</td>
<td>ARM60</td>
</tr>
<tr>
<td>ARM6</td>
<td>ARMv3</td>
<td>ARM600</td>
</tr>
<tr>
<td>ARM6</td>
<td>ARMv3</td>
<td>ARM610</td>
</tr>
<tr>
<td>ARM7</td>
<td>ARMv3</td>
<td>ARM700</td>
</tr>
<tr>
<td>ARM7</td>
<td>ARMv3</td>
<td>ARM710</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM710a</td>
</tr>
</tbody>
</table>
### ISA Versions and Part Numbers (cont)

<table>
<thead>
<tr>
<th>ARM Family</th>
<th>ARM Architecture</th>
<th>ARM Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>ARMv4T</td>
<td>ARM7TDMI(S)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM710T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM720T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM740T</td>
</tr>
<tr>
<td>ARM7EJ</td>
<td>ARMv5TEJ</td>
<td>ARM7EJ-S</td>
</tr>
<tr>
<td>ARM8</td>
<td>ARMv4</td>
<td>ARM810</td>
</tr>
<tr>
<td>StrongARM</td>
<td>ARMv4</td>
<td>SA-1</td>
</tr>
<tr>
<td>ARM9TDMI</td>
<td>ARMv4T</td>
<td>ARM9TDMI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM920T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM922T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM940T</td>
</tr>
</tbody>
</table>

### ISA Versions and Part Numbers (cont)

<table>
<thead>
<tr>
<th>ARM Family</th>
<th>ARM Architecture</th>
<th>ARM Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM9E</td>
<td>ARMv5TE</td>
<td>ARM946E-S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM966E-S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM968E-S</td>
</tr>
<tr>
<td></td>
<td>ARMv5TEJ</td>
<td>ARM926EJ-S</td>
</tr>
<tr>
<td></td>
<td>ARMv5TE</td>
<td>ARM996HS</td>
</tr>
<tr>
<td>ARM10E</td>
<td>ARMv5TE</td>
<td>ARM1020E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ARM1022E</td>
</tr>
<tr>
<td></td>
<td>ARMv5TEJ</td>
<td>ARM1026EJ-S</td>
</tr>
<tr>
<td>XScale</td>
<td>ARMv5TE</td>
<td>XScale</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bulverde</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monahans</td>
</tr>
</tbody>
</table>
ISA Versions and Part Numbers (cont)

<table>
<thead>
<tr>
<th>ARM Family</th>
<th>ARM Architecture</th>
<th>ARM Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM11</td>
<td>ARMv6</td>
<td>ARM1136J(F)-S</td>
</tr>
<tr>
<td></td>
<td>ARMv6T2</td>
<td>ARM1156T2(F)-S</td>
</tr>
<tr>
<td></td>
<td>ARMv6ZK</td>
<td>ARM1176JZ(F)-S</td>
</tr>
<tr>
<td></td>
<td>ARMv6K</td>
<td>ARM11 MPCore</td>
</tr>
<tr>
<td>Cortex-A</td>
<td>ARMv7-A</td>
<td>Cortex-A5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cortex-A8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cortex-A9 MPCore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cortex-A15 MPCore</td>
</tr>
<tr>
<td>Cortex-R</td>
<td>ARMv7-R</td>
<td>Cortex-R4(F)</td>
</tr>
<tr>
<td>Cortex-M</td>
<td>ARMv6-M</td>
<td>Cortex-M0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cortex-M1</td>
</tr>
<tr>
<td></td>
<td>ARMv7-M</td>
<td>Cortex-M3</td>
</tr>
<tr>
<td></td>
<td>ARMv7-ME</td>
<td>Cortex-M4</td>
</tr>
</tbody>
</table>

Version Timeline

XScale is a trademark of Intel Corporation
Information on Features/Applications

http://en.wikipedia.org/wiki/ARM_architecture

ARM® Processor Programmer Model
The ARM9™ processor family is built around the ARM9TDMI processor core, which implements the v4T architecture with a 5-stage pipeline, and supports the 16-bit Thumb® instruction set.

ARM920T-based ARM9 processor:
- ARM9TDMI core + Dual Caches (32K) + MMU
- Targeted for OS-based embedded applications

Separate instruction cache and data cache enabled.

Harvard cache operation:
- The AMBA bus interfaces to the rest of the system using a unified address and data buses.
ARM9TDMI and ARM7TDMI

The ARM9TDMI processor core implements a v4T ARM architecture like the ARM7TDMI

•Executes the ARM 32-bit instruction set and the compressed Thumb 16-bit instruction set

Fully code compatible to the ARM7TDMI, with two differences:

1. Better handling of the Data Abort exceptions that occurs during a memory access — Base register is restored to the original value before the exception occurs.

2. Fully implements the instruction set extension space — provides the flexibility to emulate additional instruction sets triggered through the Undefined exception.

Programmer’s Model

The Programmer’s Model describes the features of the processor available to the programmer:

• What are the registers available
• How codes are stored in the memory
• How different data types are handled
• What instructions are available to manipulate the processing of data

(Usually best understood through a processor instruction set presented at the lowest level, i.e., assembly language)
The ARM is a 32-bit architecture.

When used in relation to the ARM:
- **Nybble** means 4 bits (Halfbyte-one Hex digit)
- **Byte** means 8 bits
- **Halfword** means 16 bits
- **Word** means 32 bits (four bytes, Word definition varies for diff. processors, eg. Intel 8088 is 8, 8086 is 16, 80486 is 32, Coldfire is 16)

Most ARM’s implement two instruction sets
- 32-bit ARM Instruction Set
- 16-bit Thumb Instruction Set

Jazelle cores can also execute Java bytecode

---

The 32-bit v4T ARM processors access memory in word aligned format
- Stored in groups of four bytes in an ascending memory order
  - The first data will correspond from byte 0 to byte 3 in the memory
  - The second data will be stored from byte 4 to byte 7

Data can be in the endian format (i.e., bi-endian)

<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Word at address A</th>
<th>Halfword at address A+2</th>
<th>Halfword at address A+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte at address A+3</td>
<td>Byte at address A+2</td>
<td>Byte at address A+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Addresses used for a little-endian word
Data Types

The 32-bit v4T ARM processors support data types of the following sizes with the proper boundary alignment in the memory:

• 32-bit words
  – Aligned to four byte boundaries, with the lowest two address bits equal to zero (xxxxxx00b)

• 16-bit half-words
  – Aligned to two byte boundaries, with the lowest address bit equal to zero (xxxxxxx0b)

• 8-bit bytes
  – Can be placed on any byte boundary

Processor Operating States

The v4T based ARM processor has two operating states (don’t confuse states with modes):

1. ARM state
   – Executes 32-bit word aligned ARM instructions

2. Thumb state
   – Executes 16-bit half-word aligned Thumb instructions

The two states can be switched through the BX (Branch and Exchange) instruction.

As all exception handlings are performed in the ARM state, processors in the Thumb state will change to the ARM state when any exception occurs and revert back to the Thumb state on return.
Processor Operating Modes

The v4T-based ARM processor has seven operating modes that are further classified into Non-Privileged (1) mode and Privileged mode (6).

1. Non-Privileged mode
   - User

2. Privileged modes
   - System
   - Supervisor
   - FIQ
   - IRQ
   - Abort
   - Undefine

Processor Operating Modes Explained

- The ARM has seven basic operating modes:
  - User: unprivileged mode under which most tasks run
  - FIQ: entered when a high priority (fast) interrupt is raised (eg. fast response-battery drained-save register content to memory)
  - IRQ: entered when a low priority (normal) interrupt is raised (eg. slower response-user has pressed a button)
  - Supervisor: entered on reset and when a Software Interrupt instruction is executed
  - Abort: used to handle memory access violations (violations cause a type of interrupt sometimes called a “trap” or “exception”)
  - Undef: used to handle undefined instructions
  - System: privileged mode using the same registers as user mode
Processor Operating Modes (cont’ d)

1. User mode
   - The usual processor mode used when executing a program
   - Switch to the one of the other modes when exception occurs. (For example, when an IRQ interrupt occurs)

2. System mode
   - A privileged user mode; share the same register set as those of the User mode. But allows, for example, enabling and disabling of FIQ and IRQ interrupt
   - Useful for implementing nested priority interrupt system, and executing with an operating system

Processor Operating Modes (cont’ d)

3. Supervisor (SVC) mode
   - The default mode entered on reset and SWI exception
   - No restriction to the access of hardware, which is needed during boot up in order to initialize the processor and system
   - Normally, this is the mode the processor is in when an operating system is operating in its Protected mode

4. Fast Interrupt (FIQ) mode
   - Entered through the nFIQ exception
   - Use for fast interrupt response. For example, to support data transfer or channel process
Processor Operating Modes (cont’ d)

5. Interrupt (IRQ) mode
   – Entered through the lower priority interrupt
   – Use for general-purpose interrupt handling from peripheral and external signal sources

6. Abort mode
   – Entered through either Data Abort or Prefetch Abort exception
   – Triggered by invalid memory access violation (data or instruction prefetch access)

7. Undefined mode
   – Entered when an undefined instruction is encountered
   – Can be useful for the software emulation of hardware

Processor Registers

The v4T based processor has a total of 37 registers that are selectively made available depending on the operating modes and states of the processor. These include:
- 30 general-purpose registers
- Six status registers
- One program counter

The registers are arranged in an overlapped bank
- some registers are shared among different operating modes
Processor Registers

• General-purpose Registers
  – 32 bits wide (one word)
  – arranged in partially overlapping banks
    • This means, at an instant in time, a programmer sees 15 (r0 through r14), 1 or 2 status registers, and the PC (or r15)
  – Always have the same names (r0-r14, etc) but Physically different depending on the MODE
  – r3 may contain something DIFFERENT after a mode change!!!!!!

• Why “Swap” or “Bank Out” Registers?

ARM Registers by State

<table>
<thead>
<tr>
<th>System and User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>i0</td>
<td>i0</td>
</tr>
<tr>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>i1</td>
<td>i1</td>
</tr>
<tr>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>i2</td>
<td>i2</td>
</tr>
<tr>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>i3</td>
<td>i3</td>
</tr>
<tr>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>i4</td>
<td>i4</td>
</tr>
<tr>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>i5</td>
<td>i5</td>
</tr>
<tr>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>i6</td>
<td>i6</td>
</tr>
<tr>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>i7</td>
<td>i7</td>
</tr>
<tr>
<td>r8</td>
<td>r8_fq</td>
<td>r8_fq</td>
<td>r8_fq</td>
<td>i8_fq</td>
<td>i8_fq</td>
</tr>
<tr>
<td>r9</td>
<td>r9_fq</td>
<td>r9_fq</td>
<td>r9_fq</td>
<td>i9_fq</td>
<td>i9_fq</td>
</tr>
<tr>
<td>r10</td>
<td>r10_fq</td>
<td>r10_fq</td>
<td>r10_fq</td>
<td>i10_fq</td>
<td>i10_fq</td>
</tr>
<tr>
<td>r11</td>
<td>r11_fq</td>
<td>r11_fq</td>
<td>r11_fq</td>
<td>i11_fq</td>
<td>i11_fq</td>
</tr>
<tr>
<td>r12</td>
<td>r12_fq</td>
<td>r12_fq</td>
<td>r12_fq</td>
<td>i12_fq</td>
<td>i12_fq</td>
</tr>
<tr>
<td>r13</td>
<td>r13_fq</td>
<td>r13_fq</td>
<td>r13_fq</td>
<td>i13_fq</td>
<td>i13_fq</td>
</tr>
<tr>
<td>r14</td>
<td>r14_fq</td>
<td>r14_fq</td>
<td>r14_fq</td>
<td>i14_fq</td>
<td>i14_fq</td>
</tr>
<tr>
<td>r15 (PC)</td>
<td>r15 (PC)</td>
<td>r15 (PC)</td>
<td>r15 (PC)</td>
<td>i15 (PC)</td>
<td>i15 (PC)</td>
</tr>
</tbody>
</table>

= banked register
Register r0 to r15 and Special Uses

In the ARM state, 16 registers, plus one or two Status Registers, are visible in any of the operating modes.

Registers r0 to r15 are directly accessible in all the modes:
- r0 to r12: used for general purposes, holding either data or addresses
- r13: used as the Stack Pointer (SP)
- r14: typically used as the return address Link Register (LR) in subroutine and branch link operations
- r15: always used as the Program Counter (PC)

NOT FULLY ADHERING TO PURE RISC PHILOSOPHY
Program Counter (PC or r15) Register

- Keeps track of current instruction (IP in x86 architecture)
- Can be implemented differently in various architectures
- Illustrated with Fetch/Decode/Execute Pipeline:

<table>
<thead>
<tr>
<th>ARM</th>
<th>THUMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>PC-4</td>
<td>PC-2</td>
</tr>
<tr>
<td>PC-8</td>
<td>PC-4</td>
</tr>
</tbody>
</table>

When the processor is executing in ARM state:
- All instructions are 32 bits wide
- All instructions must be word aligned
- Therefore, the pc value is stored in bits [31:2] with bits [1:0] undefined (as 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
- Therefore, the pc value is stored in bits [31:1] with bit [0] undefined (as instructions cannot be byte aligned)

When the processor is executing in Jazelle state:
- All instructions are 8 bits wide
- Processor performs a word access to read 4 instructions at once
Current Program Status Register

Current Program Status Register (CPSR):
- Contain current state information of processor
- Contains the condition code flags of the most recently performed ALU operation
- Controls (enable/disable) the interrupts
- Controls (set) the processor mode

Shared among all the operating modes

Program Status Registers

- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation overflowed

- **Sticky Overflow flag - Q flag**
  - Architecture 5TE/J only
  - Indicates if saturation has occurred

- **J bit**
  - Architecture 5TE/J only
  - J = 1: Processor in Jazelle state (processor executes Java Bytecode)

- **Interrupt Disable bits.**
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.

- **T Bit**
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state

- **Mode bits**
  - Specify the processor mode
CPSR Bit Settings

**Mode bits** (M4–M0) determines the processor operating modes

- changes automatically when entering the exception privileged modes of different processors
- can be changed through software instruction when in any of the privileged mode (e.g., for nested interrupt support)

**Mode Bit settings (M4–M0)**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER mode</td>
<td>10000b</td>
</tr>
<tr>
<td>FIQ mode</td>
<td>10001b</td>
</tr>
<tr>
<td>SVC mode</td>
<td>10011b</td>
</tr>
<tr>
<td>UNDEFINE mode</td>
<td>1100b</td>
</tr>
<tr>
<td>SYS mode</td>
<td>11111b</td>
</tr>
<tr>
<td>IRQ mode</td>
<td>10010b</td>
</tr>
<tr>
<td>ABORT mode</td>
<td>10111b</td>
</tr>
</tbody>
</table>

CPSR Bit Settings (cont’ d)

**Conditional code flags** (N, Z, C, V)

- usually affected during the arithmetic and logical operations
- can also be changed by MSR and LDM instructions

**Control bits** F and I enable/disable the FIQ and IRQ

- setting the I bit disables the IRQ interrupt
- setting the F bit disables the FIQ interrupt

**State bit** T reflects the processor operating state

- when set, it indicates the processor is in the Thumb state
- when clear, it indicates the processor is in the ARM state
### Vector Table

<table>
<thead>
<tr>
<th>EXCEPTION TYPE</th>
<th>MODE</th>
<th>VECTOR ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>SVC</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Undefined Instructions</td>
<td>UNDEF</td>
<td>0x00000004</td>
</tr>
<tr>
<td>Software Interrupt (SWI)</td>
<td>SVC</td>
<td>0x00000008</td>
</tr>
<tr>
<td>Prefetch Abort (instruction fetch memory abort)</td>
<td>ABORT</td>
<td>0x0000000C</td>
</tr>
<tr>
<td>Data Abort (data access memory abort)</td>
<td>ABORT</td>
<td>0x00000010</td>
</tr>
<tr>
<td>IRQ (interrupt)</td>
<td>IRQ</td>
<td>0x00000018</td>
</tr>
<tr>
<td>FIQ (fast interrupt)</td>
<td>FIQ</td>
<td>0x0000001C</td>
</tr>
</tbody>
</table>

### Saved Program Status Registers

Saved Program Status Register (SPSR):
- Only available when operating in the privileged mode (i.e., except user mode)
- Contains the condition code flags and mode bits that allow entry to the privileged mode (i.e., during exception handling)

There are five SPSRs: one for each of the five privileged operating modes (except the system mode)
- Used to preserve the value of the CPSR when switching modes
- Since User and System mode are not entered during an exception, they have no SPSR
- Attempting to read SPSR while in User/System mode, unpredictable result occurs, writes are ignored
What happens during an Exception?

- When an exception occurs, the ARM:
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits
    - Change to ARM state
    - Change to exception mode
    - Disable interrupts (if appropriate)
  - Stores the return address in LR_<mode>
  - Sets PC to vector address
- To return, exception handler needs to:
  - Restore CPSR from SPSR_<mode>
  - Restore PC from LR_<mode>

This can only be done in ARM state.

ARM Application Procedure Call Standard (AAPCS)

ARM Application Procedure Call Standard (AAPCS)

- defines a standard way of how registers are used in a typical program that is coded with routines, functions, and procedures

Examples:

- r0 – r3: use to pass arguments and return results between routines call
- r4 – r8, r10 and r11: use for local variables of each routine
- r12: use as scratchpad register

Hence, routines must preserve contents of r4–r8, r10, and r11, usually by saving onto the stack.
### ARM Application Procedure Call Standard (AAPCS)

**Register variables Must be preserved**

- `r0` to `r3`
- `r4`
- `r5`
- `r6`
- `r7`
- `r8`
- `r9/sb`
- `r10/sl`
- `r11`

**Scratch register (corruptible)**

- `r12`

**Stack Pointer**

- `r13/sp`

**Link Register**

- `r14/lr`

**Program Counter**

- `r15/pc`

The compiler has a set of rules known as a Procedure Call Standard that determine how to pass parameters to a function (see AAPCS). CPSR flags may be corrupted by function call.

Assembler code which links with compiled code must follow the AAPCS at external interfaces.

The AAPCS is part of the new ABI for the ARM Architecture.

- Stack base
- Stack limit if software stack checking selected

- SP should always be 8-byte (2 word) aligned
- R14 can be used as a temporary once value stacked

---

### Banked Registers

Most of the registers in the ARM states are shared across different operating modes:

- The same registers are used in all modes.
- The content must be saved first if the value needs to be preserved.

Banked registers are separate physical registers that are swapped in and out when switching modes:

- Allow the immediate use of registers without saving the content first.
Banked Registers (cont’ d)

Notes:

1. Different banks of r13 and r14 registers (used as the Stacker pointer and Link register) are always swapped when the operating mode changes.

2. SPSRs are always available as banked registers.

3. FIQ mode (Fast interrupt) contains the largest banked register, which allows immediate access to ‘fresh’ r8 to r14 when entering this interrupt mode.

(Discussion: What is the advantage?)
ARM Core Pipelines

ARM7TDMI

Instruction Fetch

Thumb→ARM decompress

ARM decode

Reg Select

Reg Read

Shift

ALU

Reg Write

FETCH

DECODE

EXECUTE

ARM9TDMI

Instruction Fetch

ARM or Thumb Inst Decode

Reg Decode

Reg Read

Shift + ALU

Memory Access

Reg Write

FETCH

DECODE

EXECUTE

MEMORY

WRITE

ARM Core Pipelines (cont)

ARM10

Branch Prediction

Instruction Fetch

ARM or Thumb Inst Decode

Reg Read

Shift + ALU

Multiply

Memory Access

Multiply Add

Reg Write

FETCH

ISSUE

DECODE

EXECUTE

MEMORY

WRITE

ARM11

Fetch

MAC 1

MAC 2

MAC 3

Address

Data Cache 1

Data Cache 2

Shift

ALU

Saturate

Write back

Superscalar Architecture
The Thumb Processor State

Thumb Instruction Set

To reduce the higher memory footprint generally associated with RISC-based architecture

• ARM also provides 16-bit instructions: the Thumb instruction set

Can pack two Thumb instructions in one 32-bit memory location

• i.e., more instructions can be stored in the same memory device

Can use 16-bit data width memory device

• lowers the component cost

Hence, less memory means lower cost and lower operating power.
THUMB Instructions

The Thumb instruction set is a subset of the 32-bit ARM instruction set
• Some of the 32-bit ARM instructions are not available when operating in THUMB state.

For codes that cannot be implemented with the THUMB instructions
• Two cycles are needed to fetch a single 32-bit instruction.

Additional constraint
• Restrictions in some of the instructions when used in the Thumb state. (e.g., Branches have a shorter range)
• Some of the registers cannot be used

Thumb State Registers

The Thumb state provides a subset of the ARM-state registers:
• Eight general purpose registers r0 to r7 (also known as the low registers)
• SP register (banked in all modes)
• LR register (banked in all modes)
• PC register
• CPSR register
• SPSR (banked for the privileged modes)

The high registers r8 to r15 are generally not accessible when in the Thumb state (except for the three instructions MOV, ADD, and CMP).
Thumb-State Registers (cont’d)

Switching Between States

The ARM processor starts up in ARM state on reset

- Switching between the Thumb state and the ARM state is accomplished through the instruction

  \[ \text{BX } Rn \text{ (Branch and Exchange)} \]

  where \( Rn \) contains the destination address

When in the Thumb state

- Status bit ‘T’ of the CPSR will be set to 1
Unified Memory Address

The v4T ARM uses a unified memory space with the 32-bit address bus;
• hence, the total addressable memory space:
  \[2^{32} - 1 = 4 \text{ GB}\]

Memory and peripherals of specific functions are mapped in various addresses within the 4 GB address space;
• but certain address ranges are usually not utilized in typical implementation — indicated as RESERVED