16/32-Bit ARM-Thumb Architecture and AX Extensions

Presenter: *Prasad Kulkarni*

Course: CIS5930 Embedded Processors

Need for 16 Bit Thumb

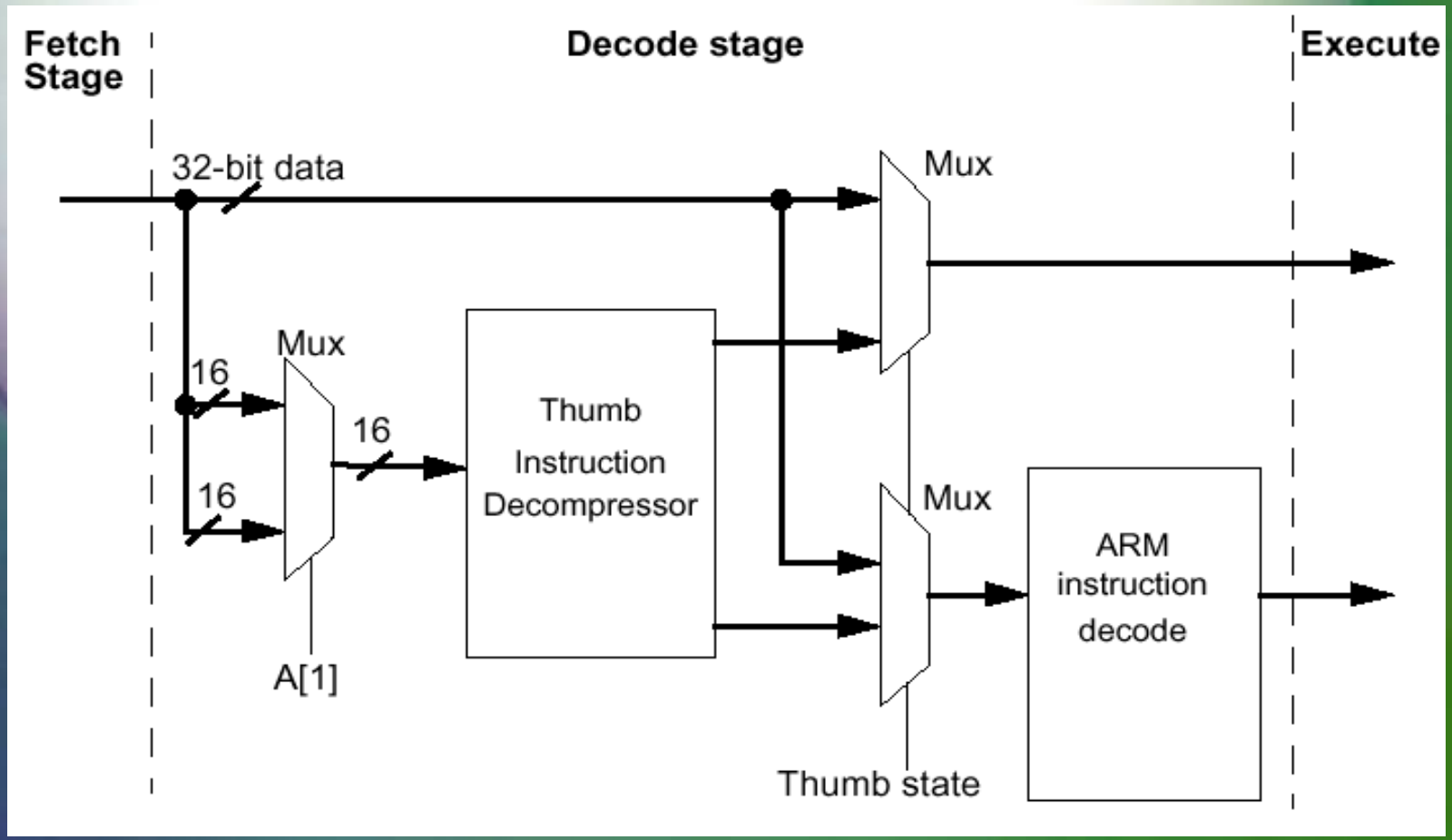
- ARM Encoding characteristics
  - fixed width 32 bit ISA
  - three address instruction set format
  - All instructions are conditional
  - 16 registers in user space
- Code size as well as power are important for embedded processors.
- ARM provides good performance with low power requirements, at the cost of code size.

Thumb ISA Characteristics

- 16 bit instruction encoding space.
- 2 address format.
- No support for predication.
- Most instructions can only access 8 registers.

Thumb Hardware

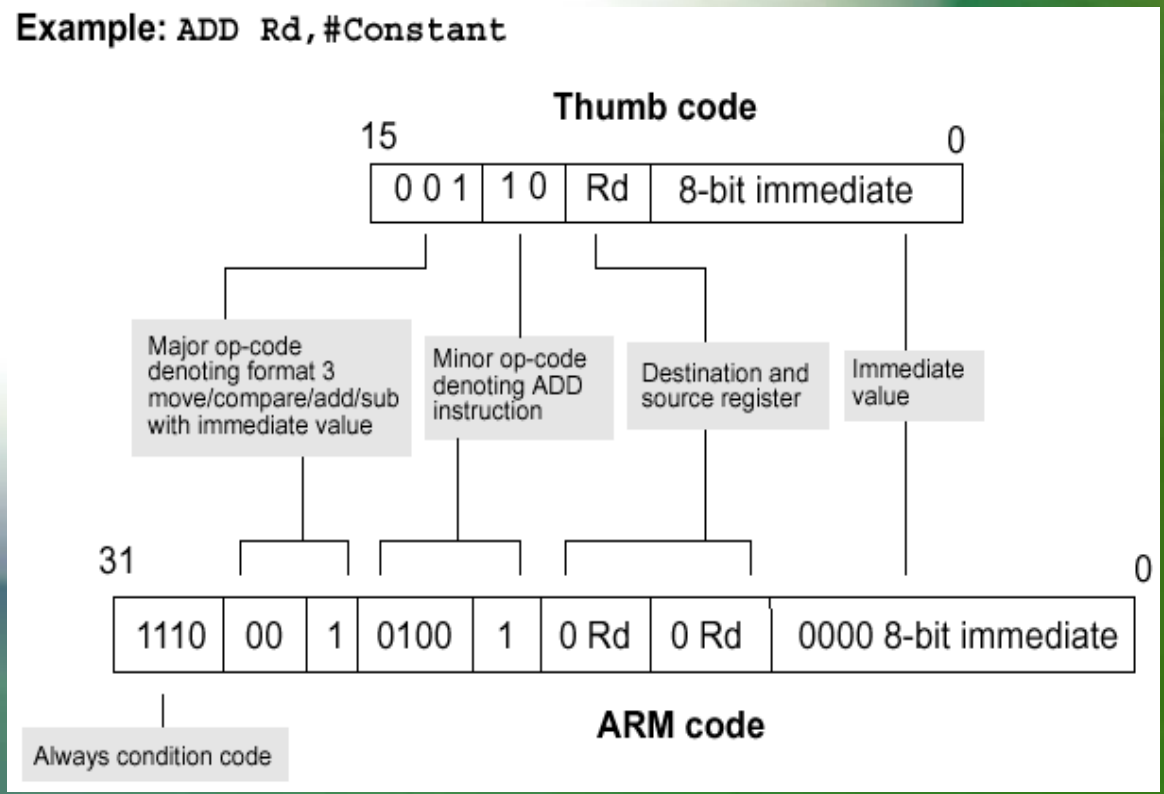
- Thumb instruction decoder is placed in the pipeline.



ARM-Thumb Instruction Mapping

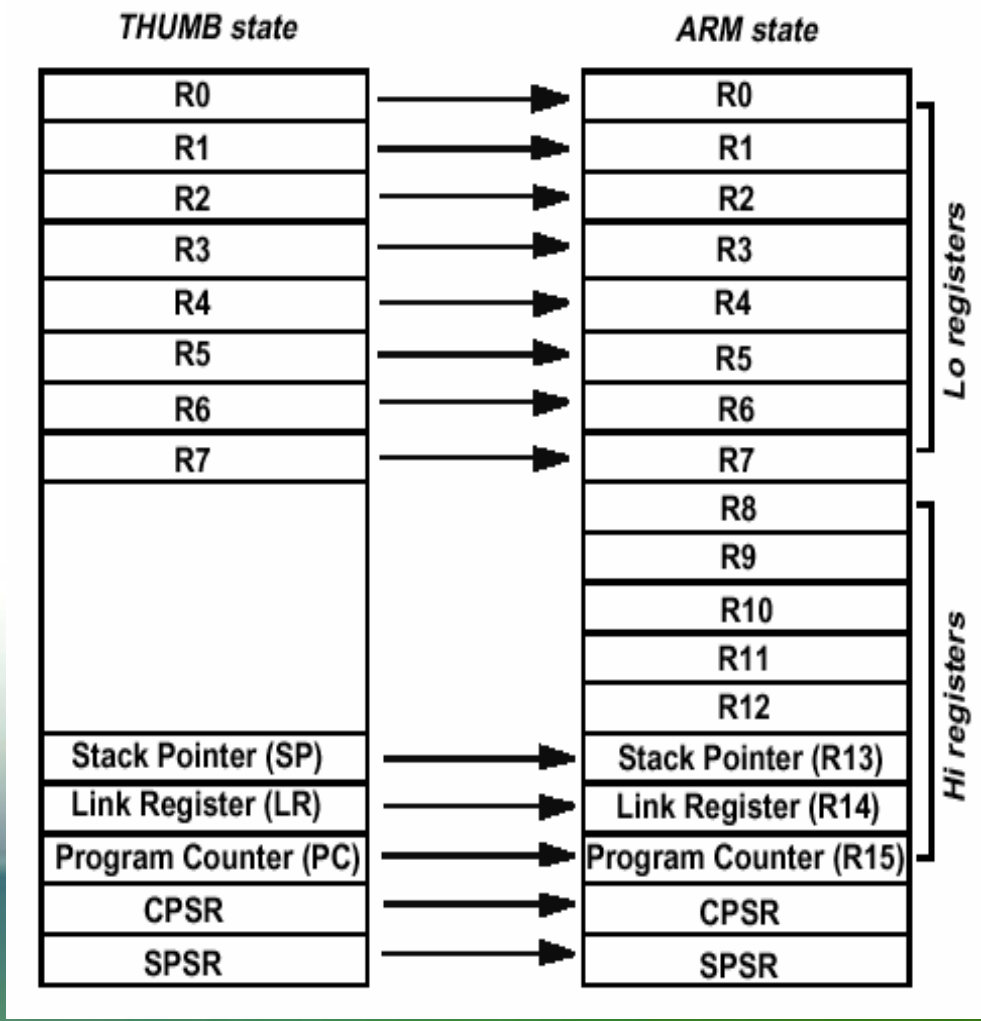
- Thumb-instruction ADD Rd, #constant is converted to unconditionally executed ARM-instruction
  ADD Rd, Rd, #constant.

- The upper register bit is fixed to zero and source and destination are equal.
- The constant is also 8-bit instead of 12-bit available in ARM-mode.

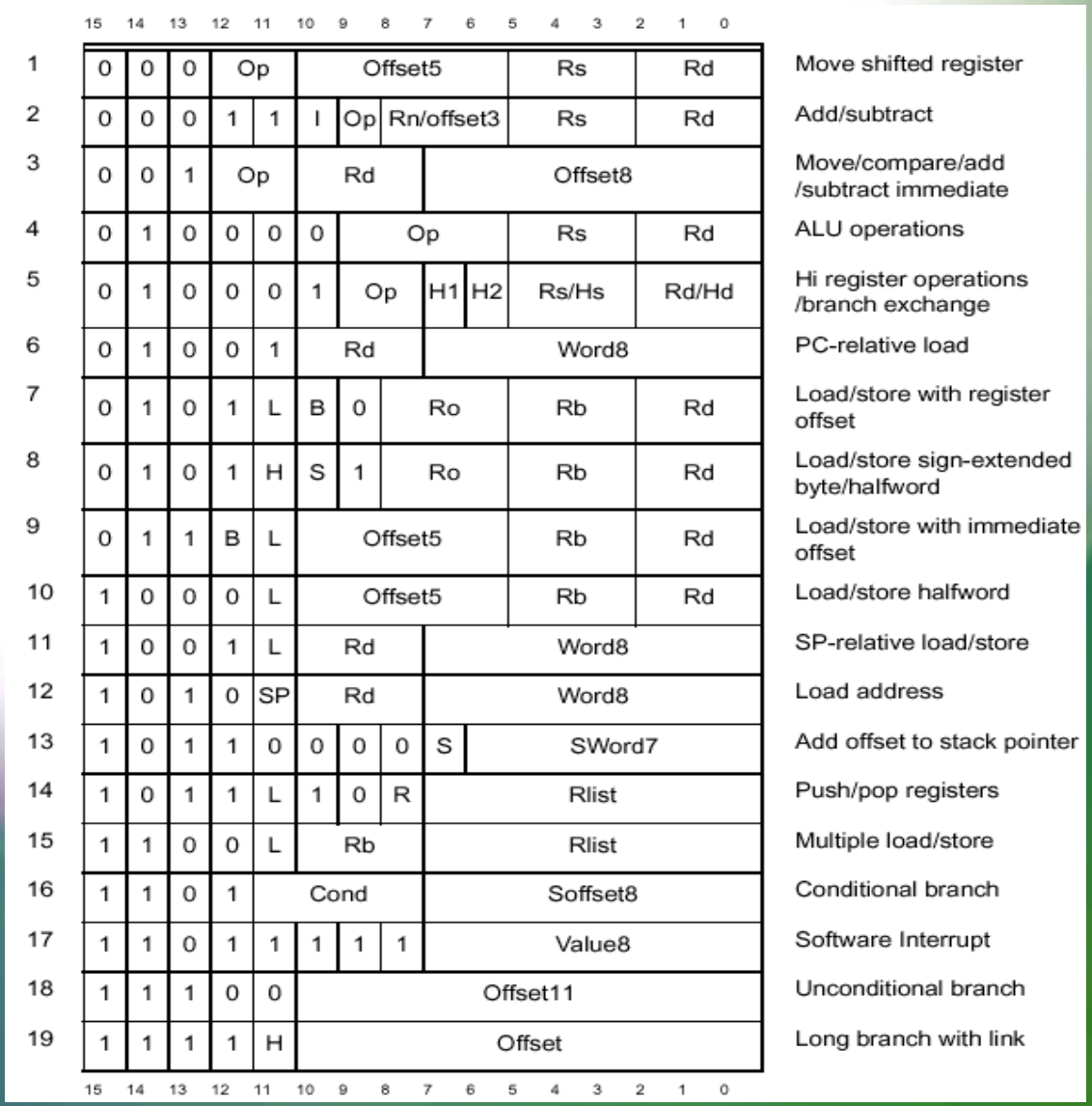


Thumb State Registers

- Only 8 of the 16 registers visible for most instructions.
- Instructions MOV, ADD, and CMP are available between register sets.



Thumb Instruction Format



Entering Thumb State

- Thumb execution is normally entered by executing an ARM BX instruction (Branch and Exchange).
- Address held in a general purpose register.
- Bit 0 of that register must be set to 1.
- BLX, LDR/LDM instructions that load the PC can also be used.
- T bit (bit 5) in the CPSR is set to 1.

Mixed ARM-Thumb Code (Example)

```
.code 16                    ;Thumb instructions folow
...
.align 2                    ; make bx word aligned
bx r15                      ; switch to ARM, r15[0] not set
nop                         ; ensure ARM code is word aligned
.code 32                    ; ARM code follows
<Shorter ARM sequence>
orr r15, r15, #1            ; set r15[0]
bx r15                      ; switch to Thumb as r15[0] is set
.code 16                    ; Thumb instructions follow
...
```

Advantages of Thumb ISA

- The main benefit of using Thumb code over ARM is reduction in code size.
- Code size reduction is reported to be 30% on average.
- In some cases, Thumb code may also achieve better performance due to fewer I-cache misses.
- Savings in power due to fewer I-cache fetches.

Issues with Thumb

- Significant increase in instruction counts, reported to be from 3% to 98%.
- Higher cycle counts for the Thumb code as compared to ARM.
- Savings in I-cache energy offset by energy used in other parts of the processor due to increase in dynamic cycles.
- User required to “blend” instruction sets by writing performance critical section of code in ARM and the rest in Thumb.

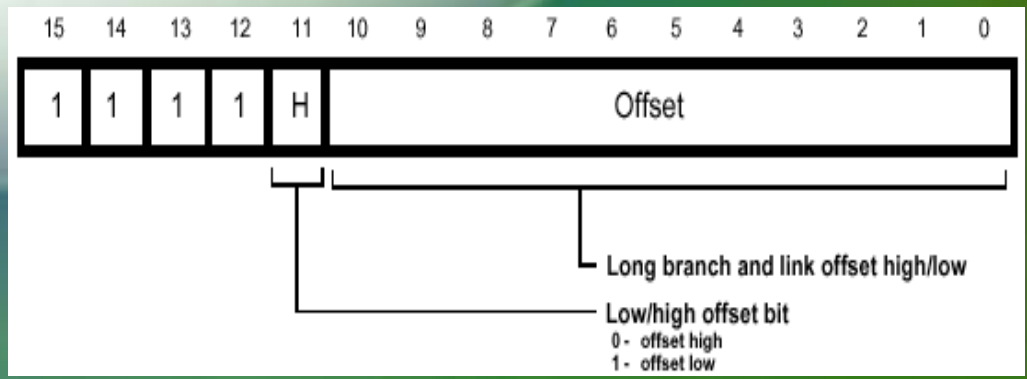
Thumb-2 Architecture

- Combined 32 and 16 bit instruction set
  - Instructions can be freely mixed
  - 16 bit instructions include the original Thumb instruction set
  - Complete compatibility with Thumb binaries
  - Some new 16 bit instructions for key code size wins
- Virtually all instructions available in ARM ISA available in Thumb-2
  - Some minor cleaning up of system management instructions
  - In principle can stand-alone as a complete ISA
- Unified assembly language for ARM and Thumb-2
  - Assembly can be targeted to either ISA

Thumb-2 Encoding

Half Word1 [15:13]	Half Word2 [12:11]	Length	Functionality
Not 111	xx	16 bits	Current 16 bit Thumb instruction
111	00	16 bits	Current 16 bit Thumb unconditional br.
111	Not 00	32 bits	Thumb-2 32 bit Thumb instruction

- Thumb-2 encodings are compatible with existing Thumb BL/BLX instructions.
- BL/BLX are 32 bit, in 2 half words.
- HW-1 (H=0) contains bits 0-11, and HW-2 contains bits 12:21 of target address.



Thumb-2 Instructions

- Two 32-bit instructions to load a 32-bit constant.

```
MOVW   Rd,#imm16  →    Rd = ZeroExtend( imm16 )
MOVT   Rd,#imm16  →    Rd[31:16] = imm16
                       // Rd[15:0] unaffected
```

- Optimizes for the common case of branch-if-zero.

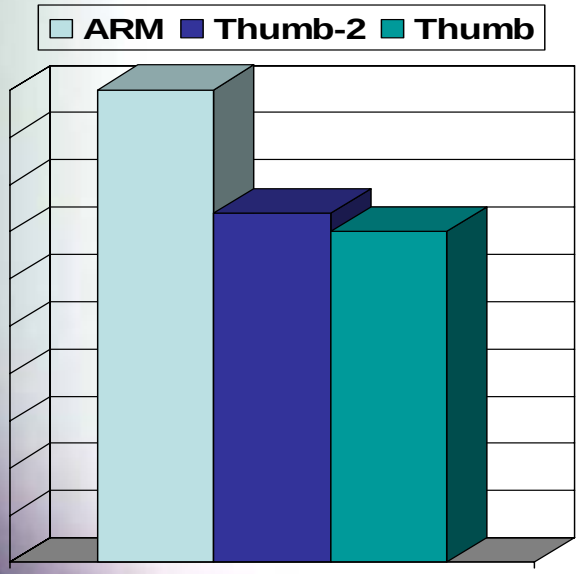
ARM	Thumb-2	Thumb
CMP r0, #0	CZB r0, ln	CMP r0, #0
BEQ ln		BEQ ln
8 Bytes, 1 or 2 cycles	**2 Bytes, 1 cycle**	**4 Bytes,1 or 2 cycles**

Thumb-2 Support for Predication

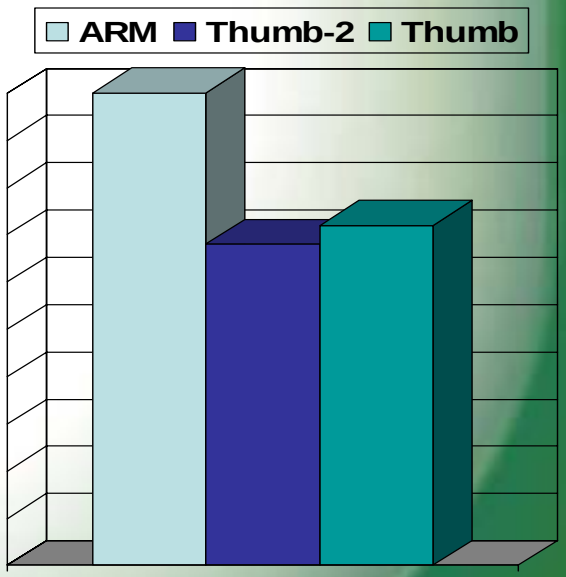
- The If-Then (IT) instruction causes the next 1-4 instructions in memory to be conditional
- Allows short conditional execution bursts in 16-bit instruction set

ARM	Thumb-2	Thumb
LDREQ r0, [r1]	ITETE EQ	BNE l1
LDRNE r0, [r2]	LDR r0, [r1]	LDR r0,[r1]
ADDEQ r0, r3, r0	LDR r0, [r2]	ADD r0, r3, r0
ADDNE r0, r4, r0	ADD r0, r3, r0	B l2
	ADD r0, r4, r0	l1: LDR r0,[r2]
		ADD r0, r4, r0
		l2 ……
16 Bytes, 4 cycles	**10 Bytes, 4 or 5 cycles**	**12 Bytes, 4 to 20 cycles**

Thumb-2 Compiled Code Size



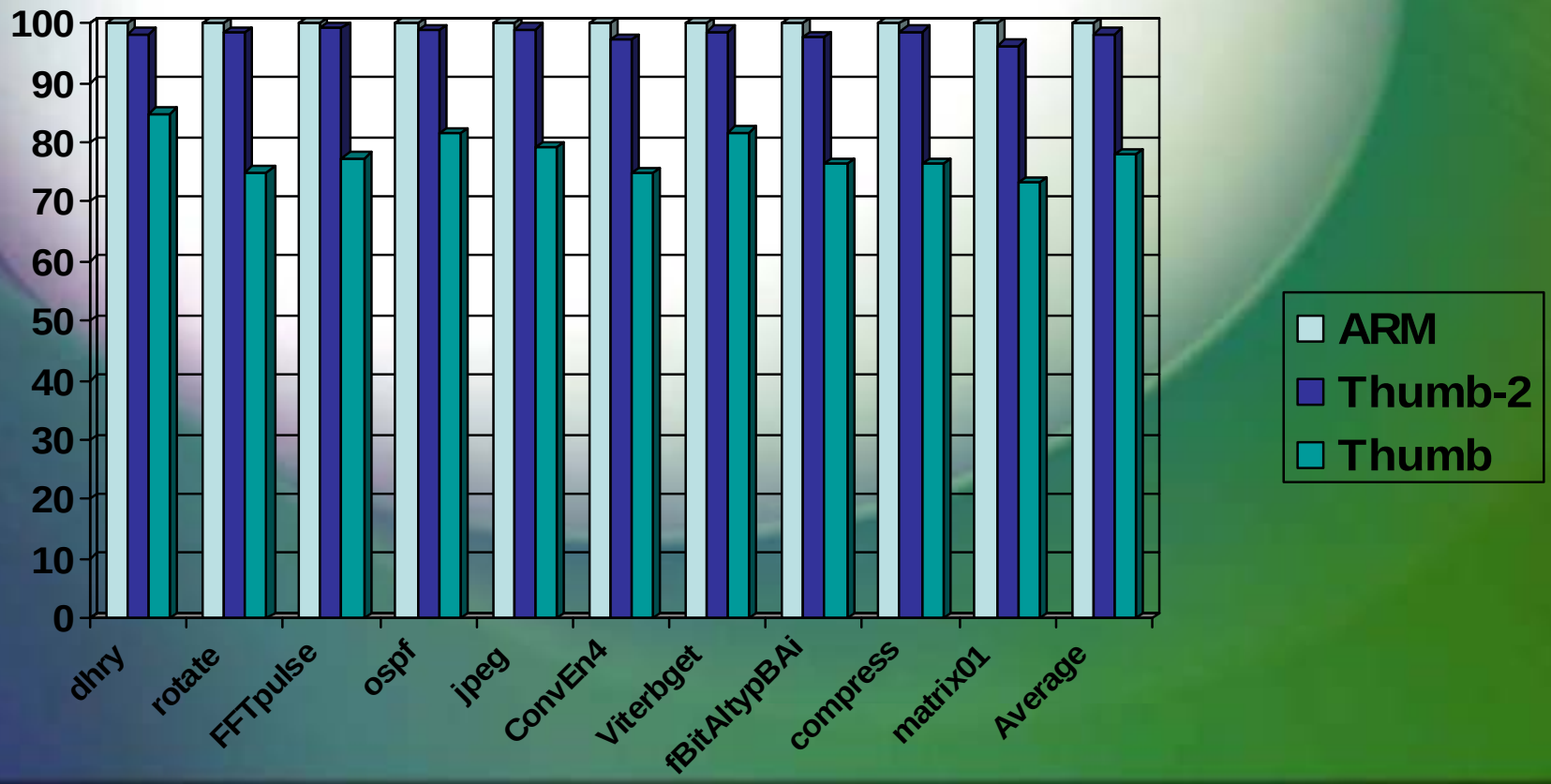
- Thumb-2 performance optimized, 26% smaller than ARM



- Thumb-2 space optimized, 32% smaller than ARM

Thumb-2 Performance

- EEMBC benchmarks, on ARM11-like core.
- Thumb-2 performance is 98% of ARM performance, and 125% of Thumb performance.



AX Extensions

- Work at the University of Arizona exploits the excess fetch bandwidth in Thumb mode to insert *augmenting* instructions.
- AX instructions are non-executing instructions that do not contribute to execution time.
- AX instruction is coalesced with following Thumb instruction at decode time.
- Reduces the instruction counts in Thumb mode by inserting AX instructions for branches, ALU operations, and MOVs.
- 8 types of AX instructions.

AX Instruction 1

- setshift
  - sets the shift type and amount for the next instruction

ARM	AXThumb
sub reg1, reg2, lsl #2	**setshift lsl, #2** sub reg1, reg2
Thumb	**Coalesced ARM**
lsl rtmp, reg2, #2 sub reg1, rtmp	sub reg1, reg2, lsl #2

AX Instruction 2

- setpred - support for predication in 16 bit code

ARM	AXThumb

cmp r3, #0
addeq r6, r6, r1
addeq r5, r5, r2
rsbne r6, r6, r1
rsbne r5, r5, r2
mov r3, r9
	(1) cmp r3, #0
	(2) setpred EQ #2
	(3) add r6, r1
	(4) sub r6, r1
	(5) add r5, r2
	(6) sub r5, r2
	(7) mov r3, r9

Thumb	**Coalesced ARM**

(1) cmp r3, #0
(2) beq (6)
(3) sub r6, r1
(4) sub r5, r2
(5) b (8)
(6) add r6, r1
(7) add r5, r2
(8) mov r3, r9
	cmp r3, #0
	sub r6, r6, r1
	sub r5, r5, r2
	mov r3, r9
	OR
	cmp r3, #0
	add r6, r6, r1
	add r5, r5, r2
	mov r3, r9

AX Instruction 3

- setsbit – set the ‘S’ bit to avoid explicit *cmp* instruction

ARM	AXThumb
movs reg1, reg2	**setsbit** mov reg1, reg2
Thumb	Coalesced ARM
mov reg1, reg2 cmp reg1, #0	movs reg1, reg2

AX Instruction 4

- setsource – sets the source register for the next instruction

ARM	AXThumb
ldr r5, [r9, #100]	**setsource high r9** ldr r5, [-, #100]
Thumb	**Coalesced ARM**
mov r3, r9 ldr r5, [r3, #100]	ldr r5, [r9, #100]

AX Instruction 5

- setdest – sets the destination register for the next instruction

ARM	AXThumb
add Hreg1, Hreg1, #imm	**setdest Hreg1** add -, #imm
Thumb	Coalesced ARM
mov rtmp, #imm add Hreg1, rtmp	add Hreg1, Hreg1, #imm

AX Instruction 6

- setthird – set the third operand (support 3-address format)

ARM	AXThumb
add reg1, reg2, reg3	**setthird reg3** add reg1, reg2
Thumb	Coalesced ARM
mov reg1, reg2 add reg1, reg3	add reg1, reg2, reg3

AX Instruction 7

- setimm – sets the immediate value for the next instruction

ARM	AXThumb
and reg1, reg1, #imm	**setimm #imm** and reg1, -
Thumb	Coalesced ARM
mov rtmp, #imm and reg1, rtmp	and reg1, reg2, #imm

AX Instruction 8

- setallhigh – indicates next instruction uses all high registers

ARM	AXThumb
push {r4,...,r11}	(1) push [r4, r5, r6, r7] (2,3) **setallhigh** push [r4, r5, r6, r7]
Thumb	**Coalesced ARM**
(1) push [r4, r5, r6, r7] (2) mov r4, r8 (3) mov r5, r9 (4) mov r6, r10 (5) mov r7, r11 (6) push [r4, r5. r6, r7]	push {r4, r5, r6, r7} push {r8, r9, r10, r11}

AX Performance Improvements

- Use of AX instructions reduces the dynamic instruction count by 10% on average over Thumb code.
- Use of AX instructions reduces cycle counts by -0.2% to 20% compared to Thumb code.
- Code size is almost identical to Thumb code.

AX Extension to Use Invisible Registers

- Another reason for the loss in performance of Thumb code as compared to ARM is reduction in the number of visible registers.
- The registers are divided into 2 sets.
- The register set currently visible to the program can be set using *setmask.*
- The compiler has to allocate and assign registers and then generate minimal number of *setmask* instructions.

Improved AX Performance

- Negligible code size increase over original Thumb code.
- Reduction in dynamic instruction count of up to 19% over Thumb, due to removal of *mov* instructions.
- Speedup of up to 20% over Thumb code.

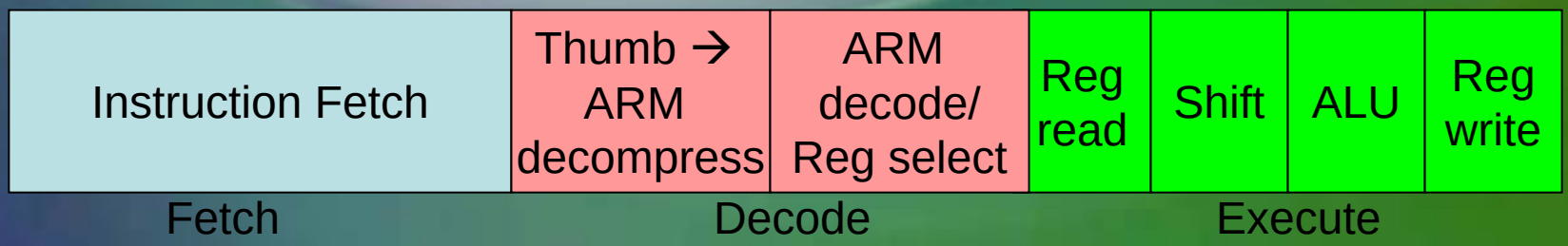
ARM Processor Families Supporting Thumb

- ARM processors provide cores for
  - application cores for platforms running complex operating system, for wireless, imaging, and consumer applications.
  - embedded cores for real-time systems for mass storage, automotive, industrial, and networking applications
  - secure cores for secure applications including smart cards, and SIMs
- Example embedded cores
  - ARM7TDMI, ARM946E-S, ARM1026EJ-S, ARM1156T2, ARM Cortex

Questions ?

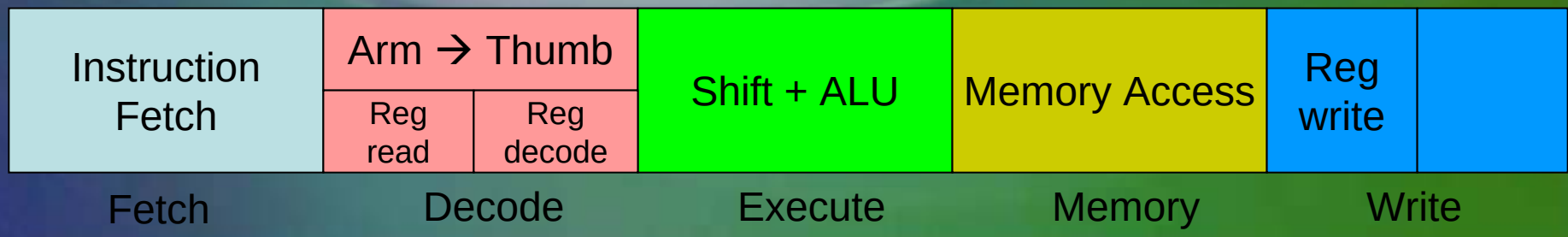
ARMv4T Architecture (ARM7TDMI)

- Introduced 16 bit Thumb ISA, von Neumann design.
- 3 stage pipeline, max 150 MHz.
- No branch prediction, 3 cycles for taken branch, 1 cycle for not-taken branch.
- Loads and stores can take multiple cycles in the Execute stage.



ARMv5TEJ Architecture (ARM926EJ)

- Better interworking between ARM and Thumb, Harvard architecture.
- 5 stage pipeline, max 233 MHz.
- Branches have similar access.
- Loads and stores now have 3 pipeline stages to finish execution.



ARMv6 Architecture (ARM1136JF)

ARMv7 Architecture