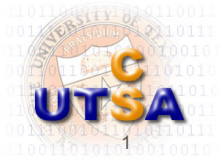
CS4833: Embedded Systems

Instruction Set Architecture (ISA) and ARM processor



Reviews

- Basic components: hardware
  - Transistors and gates
  - Combinational and sequential logic circuits
  - Timers, counters etc.
- General hardware architectures
  - von Neumann vs. Harvard
- Processing engines in embedded systems
  - Microprocessor, microcomputer, microcontroller
  - Digital signal processors (DSPs)
  - Graphics processing units (GPUs)
- Understand hardware words
  - Data, address or instructions

Outline

- Overview of ISA: outside view
  - Types: arithmetic/logic, data transfer & flow control
  - Data transfer and address modes
- Register Transfer Language (RTL): inside view
  - Registers & register operations: read & write
  - Concept of data-path
  - Instruction cycle: fetch, decode, execute, next
- Example: a simple ISA & processor design
- Case study: ISA for ARM processors

Instruction Set for A Microprocessor

- Instruction set specifies basic operations supported by one microprocessor
  - **Outside view** of what can be done: ability
  - Operations: transfer/store data, operate data and make decisions
- Different types of instructions
  - Data transfer: move/exchange data with components
  - Arithmetic and logic: computational capabilities/functions
  - Flow of control: order of instructions to be executed

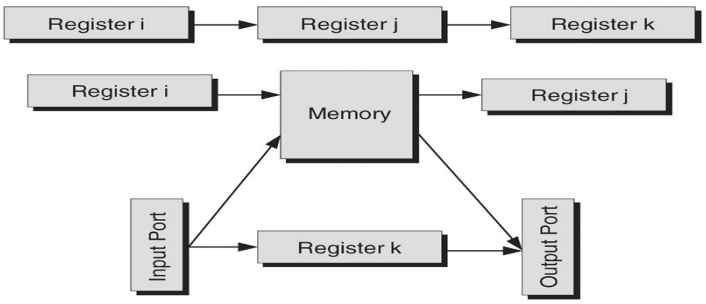
Data Transfer Instructions

- Typical data transfer instructions
  - **assembly** vs. machine codes

LD destination, source	Load—source operand transferred to destination operand can be either register or memory location.
ST source, destination	Store—source operand transferred to destination operand source must be a register and the destination must be memory.
MOVE destination, source	Transfer from register to register or memory to memory.
XCH destination, source	Interchange the source and destination operands.
PUSH/POP	Operand pushed onto or popped off of the stack.
IN/OUT destination, source	Transfer data from or to an input/output port.

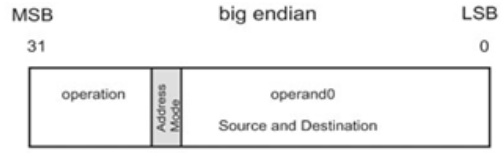
Data Transfer Instructions (cont.)

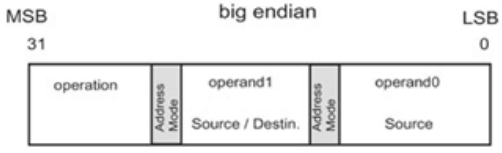
- Pieces information are needed
  - Where is the **data**: **source**
  - Where to go: **destination**
- Possible locations of data: register, memory, I/O

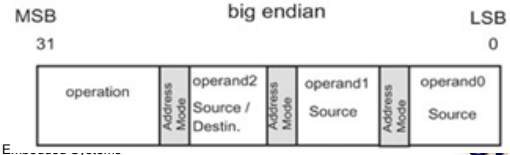


Addressing Modes: More about Operands

- Where is the data?
  - How to understand source/destination
- Addressing modes
  - Immediate
  - Direct and Indirect
  - Register direct/indirect
  - Indexed
  - Program counter relative





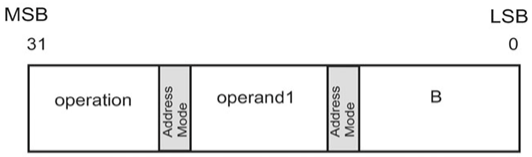


Addressing Mode: Immediate

- Typical instructions

STI	- Store immediate
LDI / LOADI	- Load Immediate
MOVI	- Move Immediate

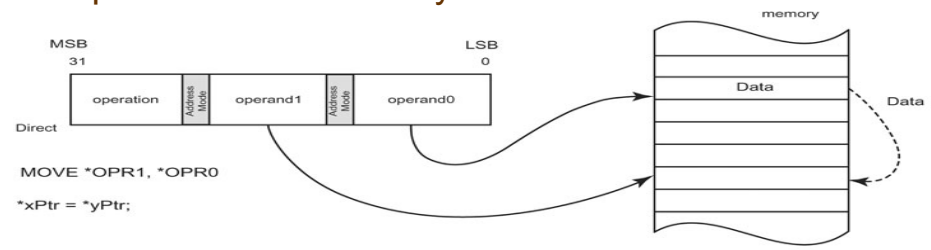
- Destination maybe implicit
- Limitation on value of immediate data

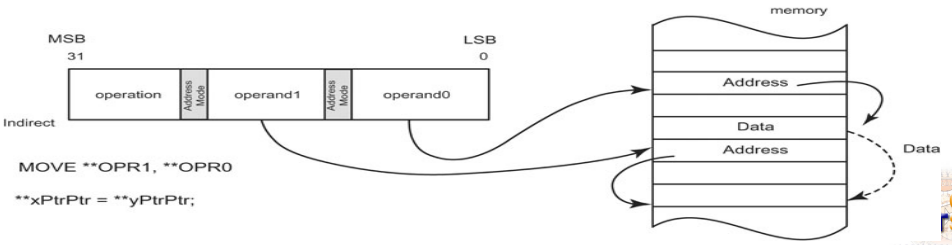



```
int x = 0xB;
MOVE OPR1, #BH
```

Addressing Mode: Direct/Indirect

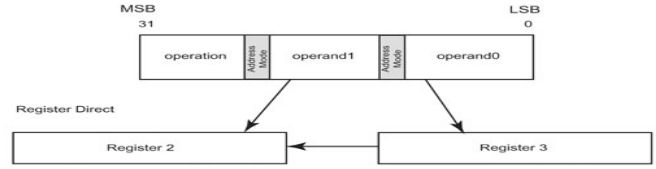
- Operand is the memory address of needed data

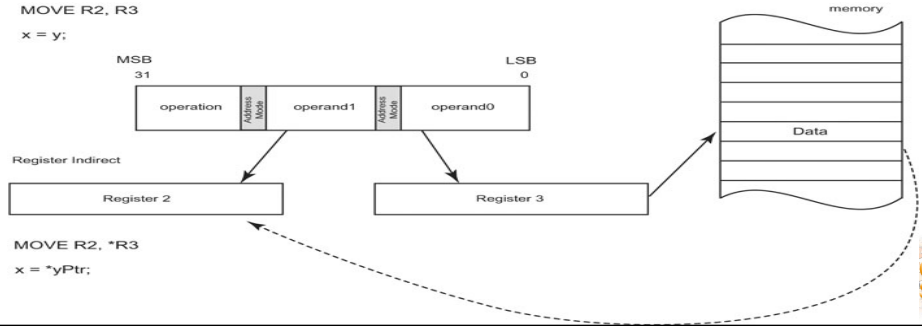




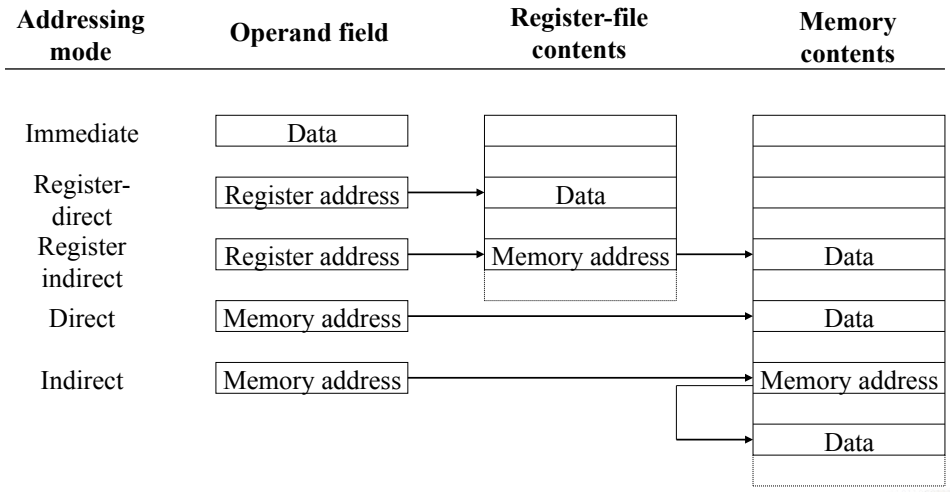
Addressing Mode: Register Direct/Indirect

- Register direct: data in register operands
- Register indirect: memory location of needed data in registers



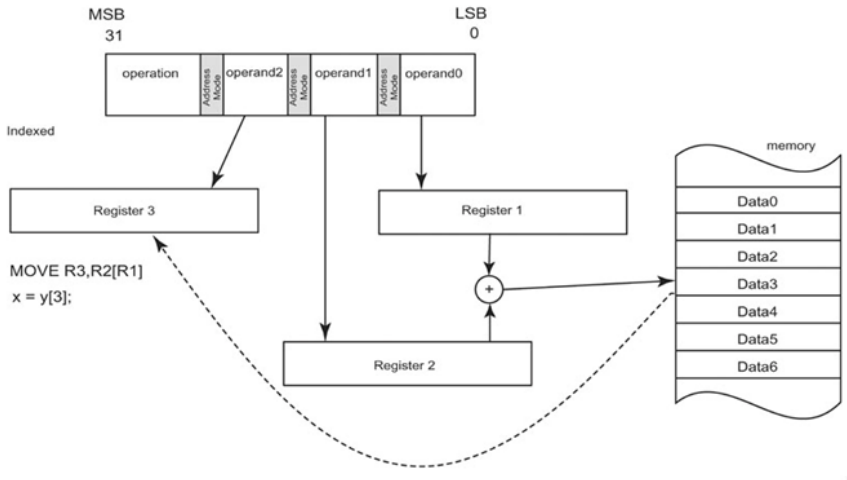


Addressing Modes



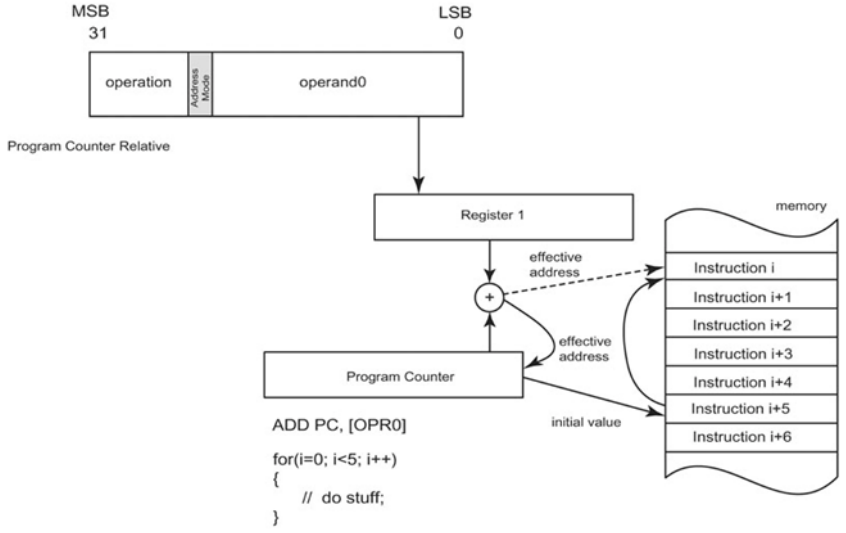
Addressing Mode: Indexed

- Base and offset of data items → access array

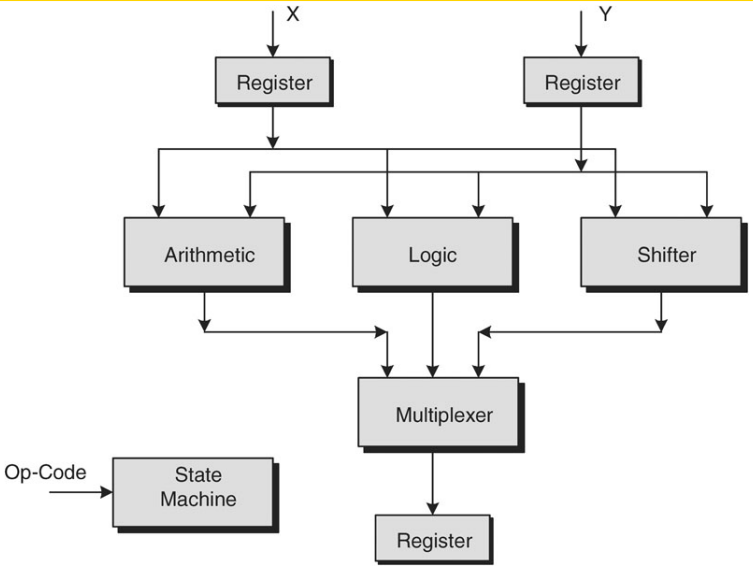


Addressing Mode: PC Relative

- For flow of control → access instructions



Arithmetic and Logic Instructions: ALU



Arithmetic and Logic Instructions

- Typical instructions

ADD2, ADD3	// Two or three operands addition
ADDC	addition with carry
SUB2, SUB3	// Two or three operands subtraction
SUBB	subtraction with borrow
MUL	multiplication
DIV	division
INC	increment
DEC	decrement
TEST	operand tested and specified condition set
TESTSET	atomic test and set

AND	bitwise AND
OR	bitwise OR
XOR	bitwise Exclusive OR
NOT or INV	complement
CLR or SET	clear or set
CLRC, SETC	carry manipulation

SHR operand, count	logical shift right
SHL operand, count	logical shift left
SHRA operand, count	arithmetic shift right
SHLA operand, count	arithmetic shift left
ROR operand, count	rotate right
ROL operand, count	rotate left

Execution Flows

- Sequential
  - Instructions are executed sequentially
- Branch
  - Decision point and instructions from different parts of program continues
- Loop
  - Repeatedly execute a set of instructions
- Procedure/function call
  - Need to leave current *running context*

Branch Instructions

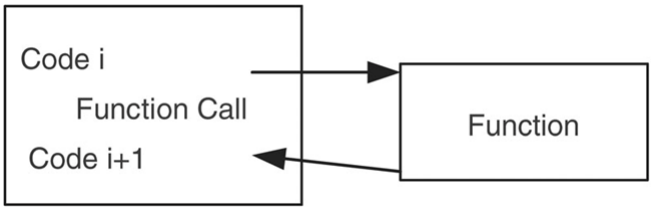
- Branch conditions and instructions

E, NE	Operand1 is *equal/not equal to* Operand2.
Z, NZ	The result of the operation is *zero/not zero.*
GT, GE	Operand1 is *greater than/greater than or equal* to Operand2.
LT, LE	Operand1 is *less than/less than or equal* to Operand2.
V	The operation resulted in an *overflow*—the result is larger than can be held in the destination.
C, NC	The operation produced a *carry/no carry.*
N	The result of the operation is *negative.*

BR label	unconditional branch to the specified label
BE label, BNE label	branch to the specified label if the equal flag is set or not set
BZ label, BNZ label	branch to the specified label if the zero flag is set or not set
BGT label	branch to the specified label if the greater than flag is set
BV label	branch to the specified label if the overflow flag is set
BC label, BNC label	branch to the specified label if the carry flag is set or not set
BN label	branch to the specified label if the negative flag is set

Procedure/Function Call

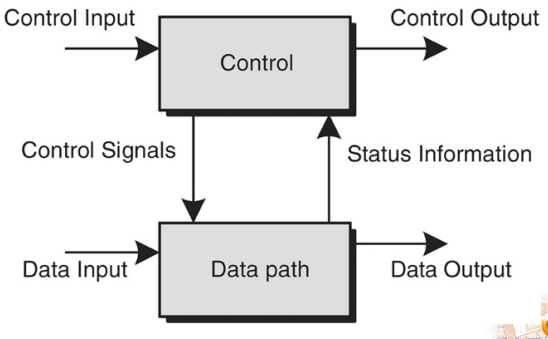
- Structure and instructions
- Running context may change: save running states



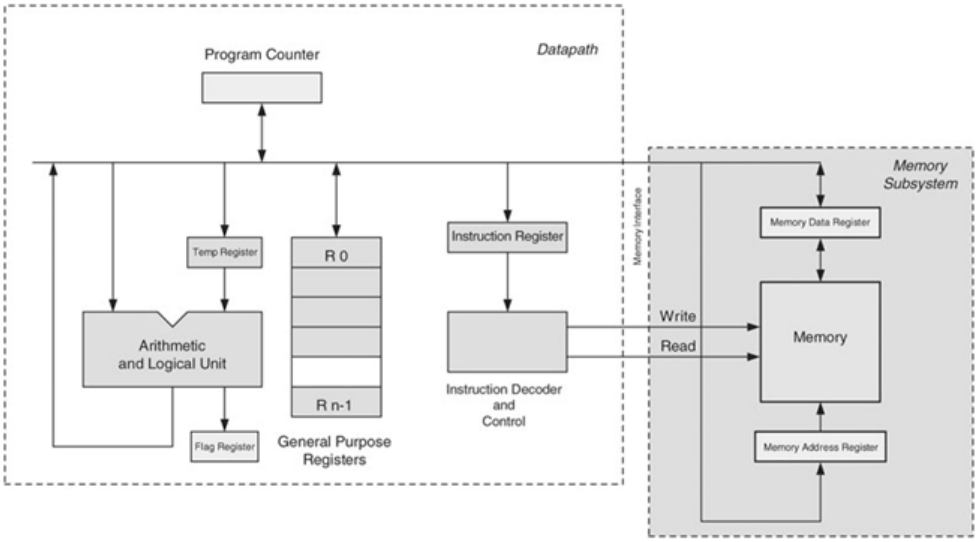
CALL operand	PC is unconditionally saved and replaced by specified operand; control is transferred to specified memory location.
RET	Previously saved contents of PC are restored, and control is returned to previous context.

Microprocessor: Register View

- **Internal view**: how things are got done (how signals are moved/changed within core **hardware**)
- Registers: containing data/address etc.
- **Control** and **data-path**
- Basic registers
  - Parallel
  - Shifted
- Register operations
  - Read
  - Write



Closer Look: Register View



Register Transfer Language (RTL)

- Describe how data is transferred: registers/memory
- Can easily translate to HDL or Verilog VHDL for implementation

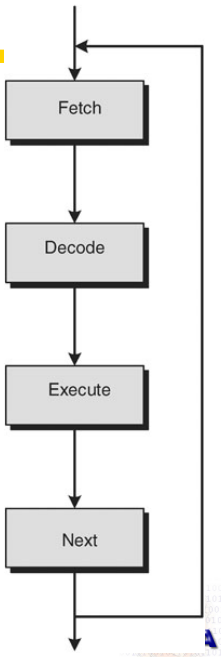
Operator	Operations
←	Transfer from right hand side to left hand side
→	If-then operation
[index]	Select word from memory at index
	… …
+-X÷	Arithmetic operation

Instruction and Register Transfer Notation

Type	Instruction	ISA level	Register Transfer
Data transfer	*Move Register*	**MOVE R1, R2**	***R1← R2***
	Move from memory	**MOVE R1, memadx**	***R1← (memadx)***
	Move to memory	**MOVE memadx, R1**	***(memadx) ← R1***
	Move immediate	**MOVE R1, #DEAD**	***R1← #DEAD***
Control Flow	*Unconditional branch*	**BR $1**	***PC← $1***
	Conditional branch	**BNE $1**	***Cond(PC←$1) if(cond) PC=$1***
Logic	*Complement accum*	**CMA**	***A← ¬ A***
	AND register	**AND R1,R2**	***R1← R1 ^ R2***
	Shift register	**SHL R1, #3**	***R1<3..0> ← R1<31-n.. 0>#(n@0)***
Arithmetic	*ADD register w. carry*	**ADDC R1, R2**	***R1←R1+R2+C***
	Clear carry	**CLRC**	***C←0***
Prog. control	*Donot execute inst.*	**NOP**	***PC←(PC+1)***
	Stop execution	**HALT**	***PC←PC***

Instruction Cycle

- Fundamental operations/steps when executing instructions
  - Fetch: retrieve instruction from memory
  - Decode: analyze op-code
  - **Mem**: get data from memory
  - Execute: perform operation
  - Next: get next instruction addr.→ PC



Register View for the Steps

- Fetch

```
MAR←PC;                 // PC enabled out to bus, MAR captures value
MDR ← Memory[MAR];      // contents of specified memory location placed into MDR
IR ← MDR;               // MDR enabled out to bus, IR captures value
```

- Execute

```
// C Instruction
      *xPtr = y;
// ISA Level Instruction
      ST *R1, R2;
// RTL Level Instructions
      MAR ← R1;             // R1 enabled out to bus, MAR captures value
      MDR ← Memory[MAR];    // contents of specified memory location placed into MDR
      R2 ← MDR;             // MDR enabled out to bus, R2 captures value
```

Example: A Simple (Trivial) Instruction Set

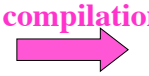
Assembly instruct.	First byte		Second byte	Operation
MOV Rn, direct	0000	Rn	direct	Rn = M(direct)
MOV direct, Rn	0001	Rn	direct	M(direct) = Rn
MOV @Rn, Rm	0010	Rn	Rm	M(Rn) = Rm
MOV Rn, #immed.	0011	Rn	immediate	Rn = immediate
ADD Rn, Rm	0100	Rn	Rm	Rn = Rn + Rm
SUB Rn, Rm	0101	Rn	Rm	Rn = Rn - Rm
JZ Rn, relative	0110	Rn	relative	PC = PC+ relative (only if Rn is 0)
	opcode		operands	

An Example of Compilation

C program

```
int total = 0;
for (int i=10; i!=0; i--)
   total += i;
// next instructions...
```

compilation

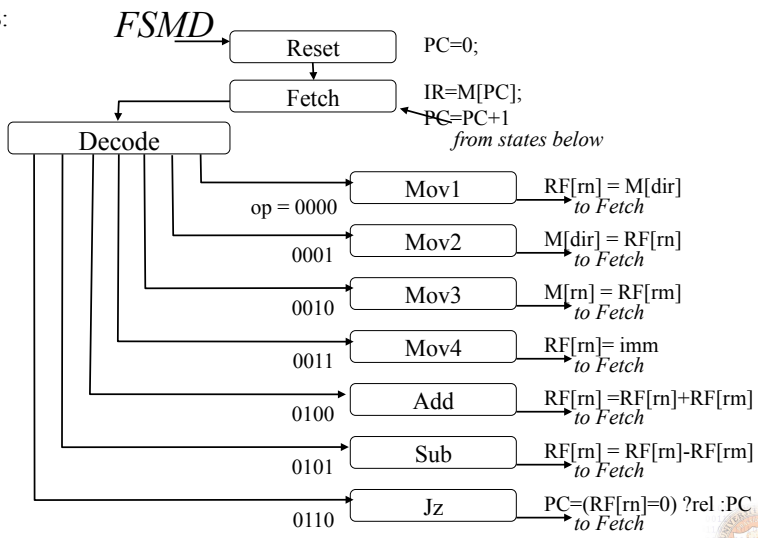


Equivalent assembly program

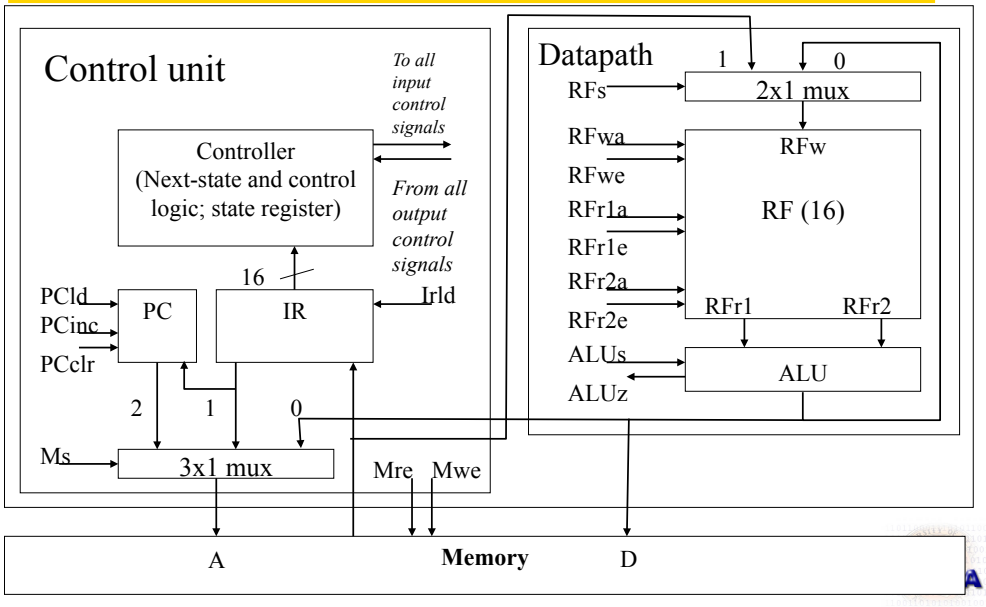
```
0       MOV R0, #0;       // total = 0
1       MOV R1, #10;      // i=10
2       MOV R2, #1;       // constant 1
3       MOV R3, #0;       // constant 0
Loop:   JZ R1, Next;      // Done if i=0
5       ADD R0, R1;       // total += i
6       SUB R1, R2;       // i--
7       JZ R3, Loop;      // Jump always
Next:   //next instruction …
```


Designing a General Purpose Processor

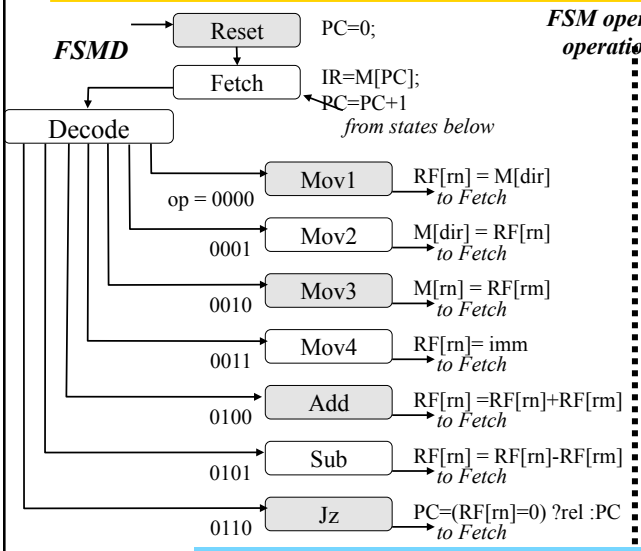
Declarations:
bit PC[16];
bit IR[16];
bit M[64k][16],
bit RF[16][16];



Architecture of a Simple Microprocessor



The Controller of the Microprocessor



FSM operations that replace the FSMD operations after a datapath is created

PCclr=1;

MS=10; Irld=1; Mre=1; PCinc=1;

RFwa=rn; RFwe=1;
RFs=01; Ms=01; Mre=1;

RFr1a=rn; RFr1e=1;Ms=01; Mwe=1;

RFr1a=rn; RFr1e=1; Ms=10; Mwe=1;

RFwa=rn; RFwe=1; RFs=10;

RFwa=rn; RFwe=1; RFs=00; RFr1a=rn;
RFr1e=1; RFr2a=rm; RFr2e=1; ALUs=00

RFwa=rn; RFwe=1; RFs=00; RFr1a=rn;
RFr1e=1; RFr2a=rm; RFr2e=1; ALUs=01

PCld= ALUz; RFrla=rn; RFrle=1;

You just built a simple microprocessor!

Case Study: ARM Instruction Set

- ARM assembly language – RISC
- ARM programming model
- ARM memory organization
- ARM data operations (32 bits)
- ARM flow of control
- Hardware-based floating point unit

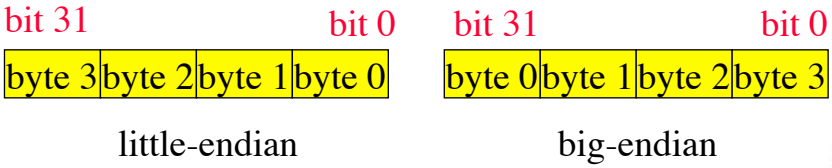
ARM Overview

- Versions of ARM
  - Several versions have been extended
  - Focus on ARMv7 and ARMv8
- Adopt standard assembly language
  - Examples: load (LDR) and ADD

```
        LDR r0,[r8] ; a comment
label   ADD r4,r0,r1
```

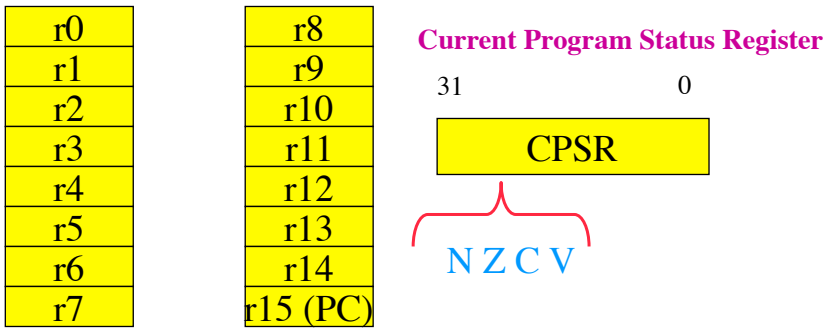

ARM Data Types

- ARM address: 32 bits long
  - Address refers to byte
- Word: 32 bits long
  - Can be divided into four 8-bit bytes
- Endianness: configured at power-up
  - Little vs. big endian mode
  - Bytes vs. endianness



ARM Programming Model

- Registers

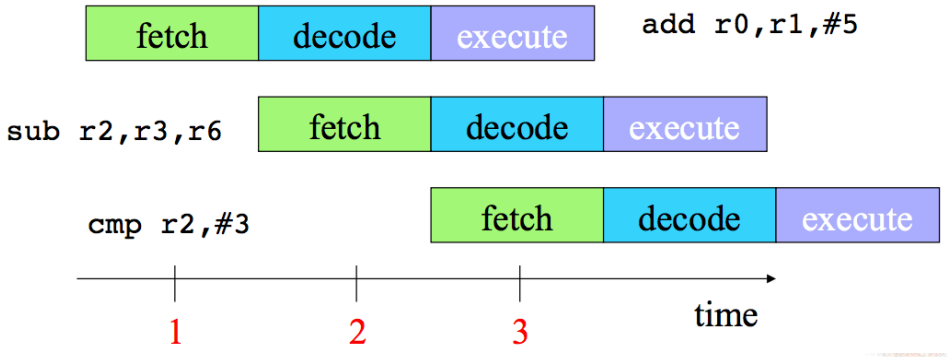


ARM Status Bits

- Every arithmetic, logical, or shifting operation sets CPSR bits:
  - N (negative)
  - Z (zero)
  - C (carry)
  - V (overflow)
- Examples: note that -1 = 0xffffffff (two's-complement)
  - -1 + 1 = 0xffffffff+0x1 = 0: NZCV = 0110.
  - $(2^{31}-1)+1$ = 0x7fffffff +0x1 = 0x80000000 = -2^{31} → NZCV = 1001.

ARM Basic Instructions and Pipeline

- Basic format: `ADD r0,r1,r2`
  - Computes r1+r2, stores in r0.
- Immediate operand: `ADD r0,r1,#5`
  - Computes r1+5, stores in r0.



ARM Data Instructions

- ADD, ADC : add (w. carry)
- SUB, SBC : subtract (w. carry)
- RSB, RSC : reverse subtract (w. carry)
- MUL, MLA : multiply (and accumulate)
- AND, ORR, EOR
- BIC : bit clear
- LSL, LSR : logical shift left/right
- ASL, ASR : arithmetic shift left/right
- ROR : rotate right
- RRX : rotate right extended with C

ARM Comparison Instructions

- CMP : compare
- CMN : negated compare
- TST : bit-wise test
- TEQ : bit-wise negated test

- These instructions set only the NZCV bits of CPSR.

ARM load/store/move Instructions

- LDR, LDRH, LDRB : load (half-word, byte)
- STR, STRH, STRB : store (half-word, byte)
- Addressing modes:
  - register indirect : LDR r0, [r1]
  - with second register : LDR r0, [r1,-r2]
  - with constant : LDR r0, [r1, #4]
- MOV, MVN : move (negated)

  MOV r0, r1 ; sets r0 to r1

Other Addressing Modes

- Base-plus-offset addressing:

```
LDR r0,[r1,#16]
```

  - Loads from location r1+16
- Auto-indexing increments base register:

```
LDR r0,[r1,#16]!
```

- Post-indexing fetches, then does offset:

```
LDR r0,[r1],#16
```

  - Loads r0 from r1, then adds 16 to r1.

Example: Assignment in C (1)

- C code: x = (a+b)-c;
- Assembler:

```
ADR r4,a      ; get address for a
LDR r0,[r4]   ; get value of a
ADR r4,b      ; get address for b, reusing r4
LDR r1,[r4]   ; get value of b
ADD r3,r0,r1  ; compute a+b
ADR r4,c      ; get address for c
LDR r2[r4]    ; get value of c
SUB r3,r3,r2  ; complete computation of x
ADR r4,x      ; get address for x
STR r3[r4]    ; store value of x
```

Example: Assignment in C (2)

- C code: z = (a<<2)|(b&15)
- Assembler:

```
ADR r4,a      ; get address for a
LDR r0,[r4]   ; get value of a
MOV r0,r0,LSL 2 ; perform shift
ADR r4,b      ; get address for b
LDR r1,[r4]   ; get value of b
AND r1,r1,#15; perform AND
ORR r1,r0,r1 ; perform OR
ADR r4,z      ; get address for z
STR r1,[r4]   ; store value for z
```


ARM Flow Control

- All operations can be performed conditionally, by testing CPSR:
  - EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE
- Branch operation: B #100
  - Can be performed conditionally

ARM subroutine linkage

- Branch and link instruction:

```
BL foo
```

  - Copies current PC to r14.
- To return from subroutine:

```
MOV r15, r14
```


ARM ISA Summary

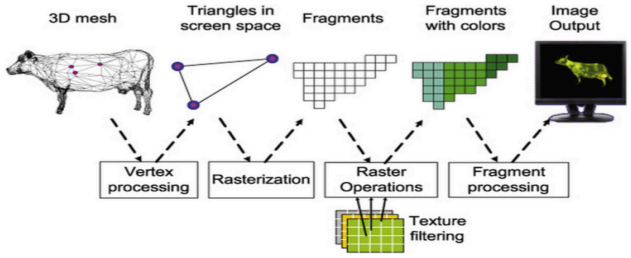
- Load/store architecture
- Most instructions are RISC
  - operate in single cycle.
  - Some multi-register operations take longer.
- All instructions can be executed conditionally
- ARMv7
  - Deployed in Cortex-A15 (Snapdragon Krait, Nvidia Tegra, TI OMAP), Cortex-A5 (AMD Fusion), Atmel microcontrollers

ARMv8: Next Generation

- Addition of 64-bit support
  - Larger virtual address space
- New instruction set (A64)
  - Fewer conditional instructions
  - New instructions to support 64-bit operands
  - No arbitrary length load/store multiple instructions
- Enhanced cryptography (both 32 and 64-bit)
- Mostly backwards compatible with ARMv7
- Expansion into higher performance markets
  - Mobile phones, servers, supercomputers
  - Cortex-A53, Apple Cyclone, Nvidia Denver

Graphics Processing Units (GPUs)

- Specialized processors for images to display
  - Primary elements: vertex and pixel shaders
- Compute Unified Device Architecture (CUDA)
  - GPGPUs: used for general purpose processing
  - Normally hundreds of simple cores
- Single program multiple data (SPMD)
  - Threading directives and memory addressing



GPU Programming

- OpenCL: general language for GPU, CPU & DSP
  - Open standard with emphasis on portability
  - Supported by Intel, AMD, Nvidia and ARM
- CUDA (Compute Unified Device Architecture)
  - Enable general purpose GPU (GPGPU)
  - Specific to NVIDA with SPMD

```
void saxpy_serial(  int n,
                    float a,
                    float *x,
                    float *y)
{
    for (int i=0: i<n; i++)
        y[i] = a*x[i] + y[i];
}

// perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

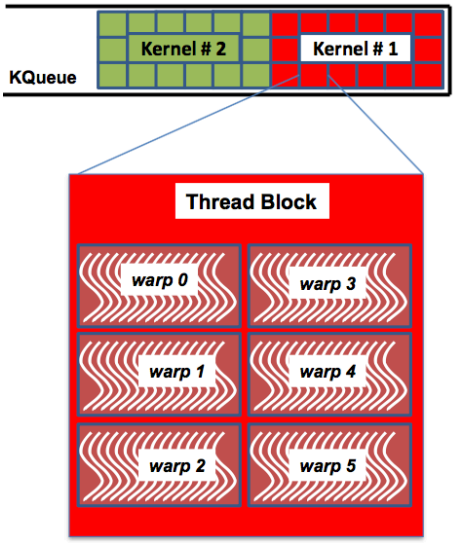
```
__global__
void saxpy_parallel(int n,
                    float a,
                    float *x,
                    float *y)
{
    int i = blockIdx.x * blockDim.x +
            threadIdx.x;

    if (i<n) y[i] = a*x[i] + y[i];
}

// perform SAXPY on 1M elements
saxpy_parallel<<<4096,256>>>(n,2.0, x, y);
```

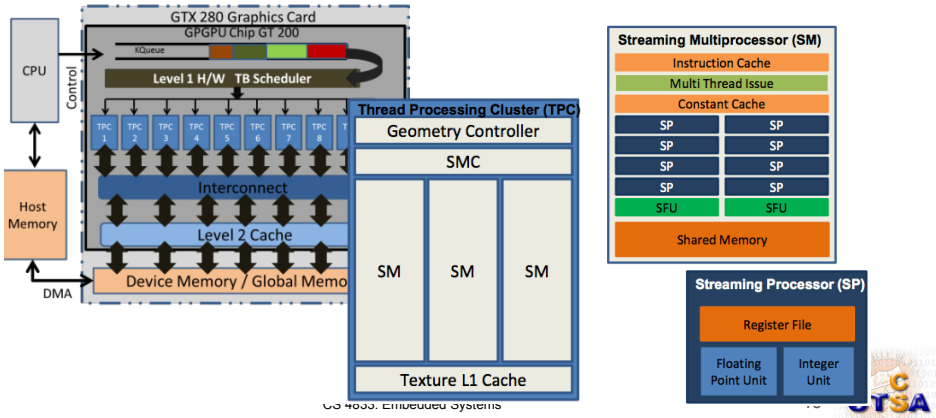

GPU Programming (cont.)

- Kernel: scales across any number of parallel cores
  - Contains grid of thread blocks
- Thread block: 1D, 2D, 3D
  - All threads in a block run the same code, over same data in shared memory space
  - Threads have ID numbers within block to select work and address shared data
  - Threads in different blocks cannot cooperate
- Threads are grouped by warps (scheduling units)



Nvidia GeForce Block Diagram

- Ultra low power (ULP) version in Tegra 3 & 4
- Desktop GTX 280 (embedded version w. diff. number of TPUs, SMs. etc.)

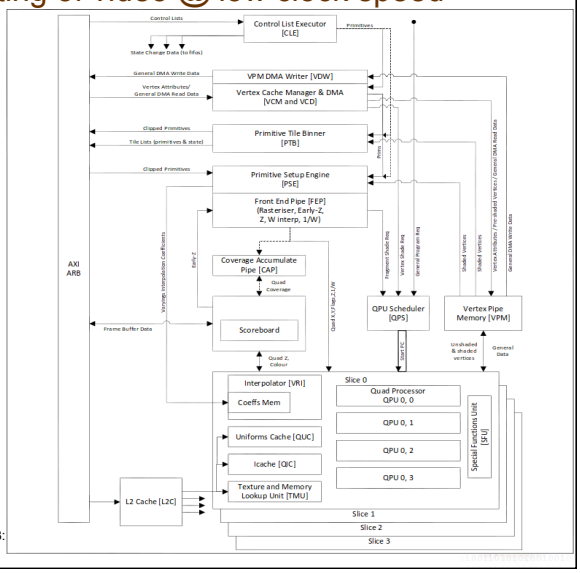


RP3: VideoCore IV GPU

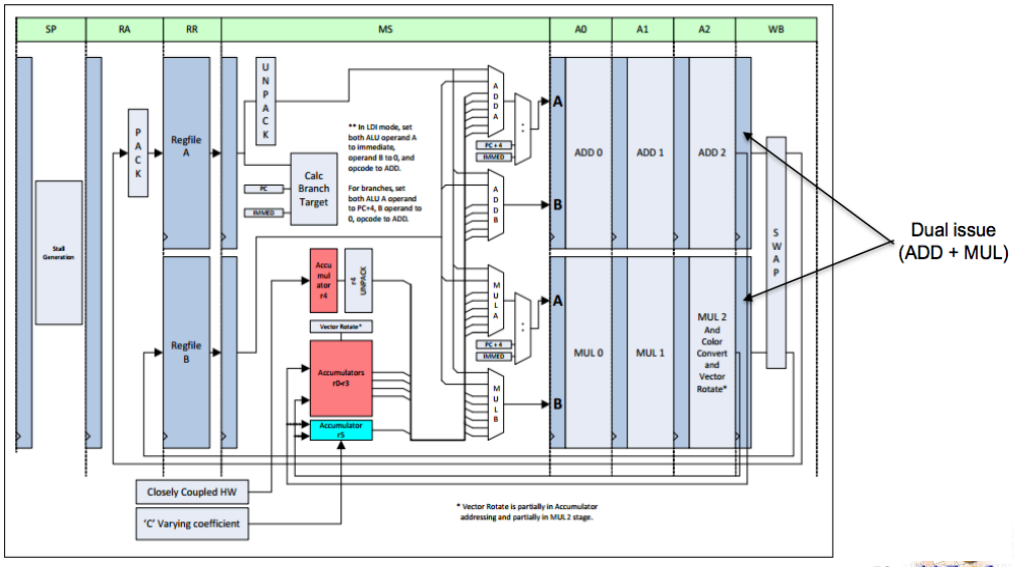
- GPU: VideoCore IV dual-core @ 400MHz
- Low-power mobile multimedia processor architecture
- Highly integrated w. CPU, memory and display
- High power efficiency in driving fast off-chip buses
- Encoder and decoder support
  - Hardware block assisting JPEG
  - 1080p30 Full HD H.264 Video
- Runs common libraries and drivers
  - OpenGL ES 2.0, OpenVG 1.1, OpenMAX
    - ✓ OpenGL: cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics

VideoCore IV GPU Architecture

- GPU core: parallel computing of video @ low clock speed
  - Co-processor extension to ARM
  - Grouped by slices: shared common cache
- Quad Processor Unit (QPU): 4-way SIMD machine
  - Four-way physical parallelism
  - 16-way virtual parallelism (4-way over four successive clock cycles)
  - RP3: 12 QPUs
- Special function Units (SPU)
  - SQRT, LOG, RECIPSQRT, EXP
- Texture and memory units
  - Specialized memory for 3D texture data



VideoCore IV GPU Pipeline: 8 Stages



Summary

- Overview of ISA: outside view
  - Types: arithmetic/logic, data transfer & flow control
  - Data transfer and address modes
- Register Transfer Language (RTL): inside view
  - Registers & register operations: read & write
  - Concept of data-path
  - Instruction cycle: fetch, decode, execute, next
- Example: a simple ISA & processor design
- Case study: ISA for ARM processors