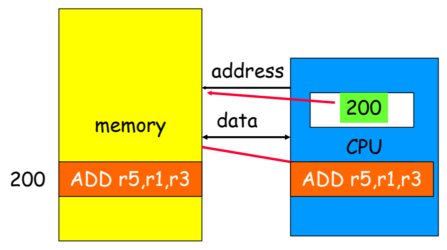
Microprocessor Architecture

- Alternative approaches
- Two opposite examples
  - SHARC
  - ARM7

Computer Architecture
in a nutshell

- Separation of CPU and memory distinguishes programmable computer.
- CPU fetches instructions from memory.
- Registers help out: program counter (PC), instruction register (IR), general-purpose registers, etc.

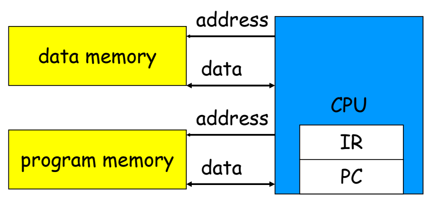
von Neumann



von Neumann vs. Harvard

- von Neumann
  - Same memory holds data, instructions.
  - A single set of address/data buses between CPU and memory
- Harvard
  - Separate memories for data and instructions.
  - Two sets of address/data buses between CPU and memory

Harvard Architecture



von Neumann vs. Harvard

- Harvard allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data:
  - greater memory bandwidth;
  - more predictable bandwidth.

RISC vs. CISC

- Reduced Instruction Set Computer (RISC)
  - Compact, uniform instructions → facilitate pipelining
  - More lines of code → large memory footprint
  - Allow effective compiler optimization
- Complex Instruction Set Computer (CISC)
  - Many addressing modes and long instructions
  - High code density
  - Often require manual optimization of assembly code for embedded systems

Microprocessors

	von Neumann	Harvard
RISC	**ARM7**	**ARM9**
CISC	**Pentium**	**SHARC (DSP)**

Digital Signal Processor = Harvard + CISC

- Streaming data
  → Need high data throughput
  → Harvard architecture
- Memory footprint
  → Require high code density
  → Need CISC instead of RISC

DSP Optimizations

- Signal processing
  - Support **floating point** operation
  - Efficient **loops** (matrix, vector operations)
  - Ex. Finite Impulse Response (FIR) filters
- Real-time requirements
  - Execution time must be predictable → opportunistic optimization in general purpose processors may not work (e.g., caching, branch prediction)

SHARC Architecture

- Modified Harvard architecture.
  - Separate data/code memories.
  - Program memory can be used to store data.
    - Two pieces of data can be loaded in parallel
- Support for signal processing
  - Powerful floating point operations
  - Efficient loop
  - Parallel instructions

Registers

- Register files
  - 40 bit R0-R15 (aliased as F0-F15 for floating point)
- Data address generator registers.
- Loop registers.
- Register files connect to:
  - multiplier
  - shifter;
  - ALU.

Assembly Language

- 1-to-1 representation of binary instructions
- Why do we need to know?
  - Performance analysis
  - Manual optimization of critical code
- Focus on architecture characteristics
  - NOT specific syntax

SHARC Assembly

- Algebraic notation terminated by semicolon:

```
       R1=DM(M0,I0), R2=PM(M8,I8); ! comment
label: R3=R1+R2;
```

data memory access program memory access

Computation

- Floating point operations
- Hardware multiplier
- Parallel computation

Data Types

- 32-bit IEEE single-precision floating-point.
- 40-bit IEEE extended-precision floating-point.
- 32-bit integers.
- 48-bit instructions.

Rounding and Saturation

- Floating-point can be:
  - Rounded toward zero;
  - Rounded toward nearest.
- ALU supports saturation arithmetic
  - Overflow results in max value, not rollover.
- CLIP Rx within range [-Ry,Ry]
  - **`Rn = CLIP Rx by Ry;`**

Parallel Operations

- Can issue some computations in parallel:
  - dual add-subtract;
  - multiplication and dual add/subtract
  - floating-point multiply and ALU operation

```
R6 = R0*R4, R9 = R8 + R12, R10 = R8 - R12;
```

Example: Exploit Parallelism

```
if (a>b) y = c-d; else y = c+d;
```

Compute both cases, then choose which one to store.

```
! Load values
  R1=DM(_a); R2=DM(_b);
  R3=DM(_c); R4=DM(_d);
! Compute both sum and difference
  R12 = R2+R4, R0 = R2-R4;
! Choose which one to save
  COMP(R1,R2);
  IF LE R0 = R12;
  DM(_y) = R0 ! Write to y
```

Memory Access

- Parallel load/store
- Circular buffer

Load/Store

- Load/store architecture
- Can use direct addressing
- Two set of data address generators (DAGs):
  - program memory;
  - data memory.
  - Can perform two load/store per cycle
- Must set up DAG registers to control loads/stores.

Basic Addressing

- Immediate value:
  `R0 = DM(0x20000000);`
- Direct load:
  `R0 = DM(_a); ! Load contents of _a`
- Direct store:
  `DM(_a)= R0; ! Stores R0 at _a`

DAG1 Registers

I0	M0	L0	B0
I1	M1	L1	B1
I2	M2	L2	B2
I3	M3	L3	B3
I4	M4	L4	B4
I5	M5	L5	B5
I6	M6	L6	B6
I7	M7	L7	B7

Post-Modify with Update

- I register holds base address.
- M register/immediate holds modifier value.

```
R0 = DM(I3,M3) ! Load
DM(I2,1) = R1 ! Store
```

Circular Buffer

- L: buffer size
- B: buffer base address
- I, M in post-modify mode
- I is automatically wrapped around the circular buffer when it reaches B+L
- Example: FIR filter

Zero-Overhead Loop

- No cost for jumping back to start of loop
  - Decrement counter, cmp, and jump back

```
     LCNTR=30, DO L UNTIL LCE;
     R0=DM(I0,M0), F2=PM(I8,M8);
     R1=R0-R15;
L:   F4=F2+F3;
```

FIR Filter on SHARC

```
   ! Init: Set up circular buffers for x[] and c[].
   B8=PM(_x); ! I8 is automatically set to _x
   L8=4; ! Buffer size
   M8=1; ! Increment of x[]
   B0=DM(_c); L0=4; M0=1; ! Set up buffer for c

   ! Executed after new sensor data is stored in xnew
   R1=DM(_xnew);
   ! Use post-increment mode
   PM(I8,M8)=R1;
   ! Loop body
   LCNTR=4, DO L UNTIL LCE;
   ! Use post-increment mode
   R1=DM(I0,M0), R2=PM(I8,M8);
L: R8=R1*R2, R12=R12+R8;
```

Nested Loop

- PC Stack
  - Loop start address
  - Return addresses for subroutines
  - Interrupt service routines
  - Max depth = 30
- Loop Address Stack
  - Loop end address
  - Max depth = 6
- Loop Counter Stack
  - Loop counter values
  - Max depth = 6

Example: Nested Loop

```
S1:   LCNTR=3, DO LP2 UNTIL LCE;
S2:   LCNTR=2, DO LP1 UNTIL LCE;
      R1=DM(I0,M0), R2=PM(I8,M8);
LP1:  R8=R1*R2;
      R12=R12+R8;
LP2:  R11=R11+R12;
```

SHARC

- CISC + Harvard architecture
- Computation
  - Floating point operations
  - Hardware multiplier
  - Parallel operations
- Memory Access
  - Parallel load/store
  - Circular buffer
- Zero-overhead and nested loop

Microprocessors

	von Neumann	Harvard
RISC	**ARM7**	**ARM9**
CISC	**Pentium**	**DSP (SHARC)**

ARM7

- von Neumann + RISC
- Compact, uniform instruction set
  - 32 bit or 12 bit
  - Usually one instruction/cycle
  - Poor code density
  - No parallel operations
- Memory access
  - No parallel access
  - No direct addressing

FIR Filter on ARM7

```
; loop initiation code
   MOV r0, #0 ; use r0 for loop counter
   MOV r8, #0 ; use separate index for arrays
   LDR r1, #4 ; buffer size
   MOV r2, #0 ; use r2 for f
   ADR r3, c ; load r3 with base of c[ ]
   ADR r5, x ; load r5 with base of x[ ]

; loop; instructions for circular buffer are not shown
L: LDR r4, [r3, r8] ; get c[i]
   LDR r6, [r5, r8] ; get x[i]
   MUL r4, r4, r6 ; compute c[i]x[i]
   ADD r2, r2, r4 ; add into sum
   ADD r8, r8, #4 ; add one word to array index
   ADD r0, r0, #1 ; add 1 to i
   CMP r0, r1 ; exit?
   BLT L ; if i < 4, continue
```

Sample Prices

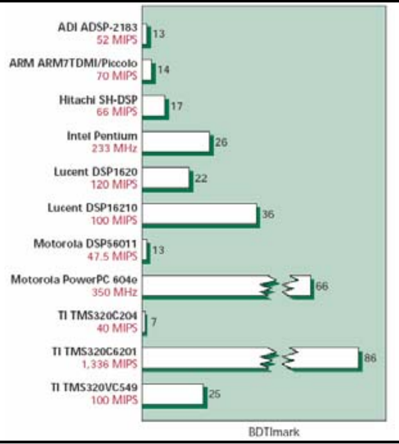
- ARM7: $14.54
- SHARC: $51.46 - $612.74

MIPS/FLOPS Metrics

- Do not indicate how much work is accomplished by each instruction.
  - Depend on architecture and instruction set.
- Especially unsuitable for DSPs due to the diversity of architecture and instruction sets.
  - Circular buffer load
  - Zero-overhead loop

Evaluating DSP Speed

- Implement, manually optimize, and compare complete application on multiple DSPs
  - Time consuming
- Benchmarks: a set of small pieces (kernel) of representative code
  - Ex. FIR filter
  - Inherent to most embedded systems
  - Small enough to allow manual optimization on multiple DSPs
- Application profile + benchmark testing
  - Assign relative importance of each kernel



Other Important Metrics

- Power consumption
- Cost
- Code density
- … …

Reading

- Chapter 2 (only the sections related to slides)
- Optional: J. Eyre and J. Bier, DSP Processors Hit the Mainstream, IEEE Micro, August 1998.
- Optional: More about SHARC
  - http://www.analog.com/processors/processors/sharc/
  - Nested loops: Pages (3-37) – (3-59) http://www.analog.com/UploadedFiles/Associated_Docs/476124543020432798236x_pgr_sequen.pdf