**CPUs**

- Input and output.
- Supervisor mode, exceptions, traps.
- Co-processors.

**I/O devices**

- Usually includes some non-digital component.
- Typical digital interface to CPU:

```
CPU
status reg
data reg
   mechanism
```

**Application: 8251 UART**

- Universal asynchronous receiver transmitter (UART) provides serial communication.
- 8251 functions are integrated into standard PC interface chip.
- Allows many communication parameters to be programmed.

**Serial communication**

- Characters are transmitted separately:

```
no char
start bit 0 bit 1 ... bit(n-1) stop
```

**Serial communication parameters**

- Baud (bit) rate.
- Number of bits per character.
- Parity/no parity.
- Even/odd parity.
- Length of stop bit (1, 1.5, 2 bits).

**8251 CPU interface**

```
Two types of instructions can support I/O:
- special-purpose I/O instructions;
- memory-mapped load/store instructions.

Intel x86 provides in, out instructions. Most other CPUs use memory-mapped I/O.

I/O instructions do not preclude memory-mapped I/O.

Define location for device:
```
DEV1 EQU 0x1000
```

Read/write code:
```
LDR r1,#DEV1 ; set up device
LDR r0,[r1] ; read DEV1
LDR r0,#8 ; set up value to write
STR r0,[r1] ; write value to device
```

Device must be in external memory space (above 0x400000).

Use DM to control access:
```
I0 = 0x400000;
M0 = 0;
R1 = DM(I0,M0);
```

Busy/wait is very inefficient.
- CPU can't do other work while testing device.
- Hard to do simultaneous I/O.

Interrupts allow a device to change the flow of control in the CPU.
- Causes subroutine call to handle device.

Based on subroutine call mechanism.
- Interrupt forces next instruction to be a subroutine call to a predetermined location.
- Return address is saved to resume executing foreground program.
**Interrupt physical interface**

- CPU and device are connected by CPU bus.
- CPU and device handshake:
  - device asserts interrupt request;
  - CPU asserts interrupt acknowledge when it can handle the interrupt.

**Example: interrupt I/O with buffers**

- Queue for characters:

```
+-------+-------+-------+-------+
|       |       |       |       |
|       |       |       |       |
|   z   |       |       |       |
|   x   |       |       |       |
| head  | tail  |       |       |
```

**I/O sequence diagram**

- Foreground program:
- Input:
- Output:
- Queue:
  - empty
  - a
  - empty
  - b
  - empty
  - c

**Debugging interrupt code**

- What if you forget to change registers?
  - Foreground program can exhibit mysterious bugs.
  - Bugs will be hard to repeat—depend on interrupt timing.

**Priorities and vectors**

- Two mechanisms allow us to make interrupts more specific:
  - Priorities determine what interrupt gets CPU first.
  - Vectors determine what code is called for each type of interrupt.
- Mechanisms are orthogonal: most CPUs provide both.

**Prioritized interrupts**

- device 1
- device 2
- device n

- Interrupt acknowledge:
  - L1 L2 L3
- CPU
**Interrupt prioritization**

- **Masking**: Interrupt with priority lower than current priority is not recognized until pending interrupt is complete.
- **Non-maskable interrupt (NMI)**: Highest-priority, never masked.
  - Often used for power-down.

**Example: Prioritized I/O**

![Diagram of prioritized I/O]

**Interrupt vectors**

- Allow different devices to be handled by different code.
- **Interrupt vector table**:
  - Interrupt vector table head
  - handler 0
  - handler 1
  - handler 2
  - handler 3

**Interrupt vector acquisition**

- CPU
- device
- receive request
- receive ack
- receive vector

**Generic interrupt mechanism**

- Assume priority selection is handled before this point.

**Interrupt sequence**

- CPU acknowledges request.
- Device sends vector.
- CPU calls handler.
- Software processes request.
- CPU restores state to foreground program.
**Sources of interrupt overhead**

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.

**ARM interrupts**

- ARM7 supports two types of interrupts:
  - Fast interrupt requests (FIQs).
  - Interrupt requests (IRQs).
- Interrupt table starts at location 0.

**ARM interrupt procedure**

- CPU actions:
  - Save PC. Copy CPSR to SPSR.
  - Force bits in CPSR to record interrupt.
  - Force PC to vector.
- Handler responsibilities:
  - Restore proper PC.
  - Restore CPSR from SPSR.
  - Clear interrupt disable flags.

**ARM interrupt latency**

- Worst-case latency to respond to interrupt is 27 cycles:
  - Two cycles to synchronize external request.
  - Up to 20 cycles to complete current instruction.
  - Three cycles for data abort.
  - Two cycles to enter interrupt handling state.

**SHARC interrupt structure**

- Interrupts are vectored and prioritized.
- Priorities are fixed: reset highest, user SW interrupt 3 lowest.
- Vectors are also fixed. Vector is offset in vector table. Table starts at 0x20000 in internal memory, 0x40000 in external memory.

**SHARC interrupt sequence**

Start: must be executing or IDLE/IDLE16.
1. Output appropriate interrupt vector address.
2. Push PC value onto PC stack.
3. Set bit in interrupt latch register.
4. Set IMASKP to current nesting state.
**SHARC interrupt return**

Initiated by RTI instruction.
1. Return to address at top of PC stack.
2. Pop PC stack.
3. Pop status stack if appropriate.
4. Clear bits in interrupt latch register and IMASKP.

**SHARC interrupt performance**

Three stages of response:
- 1 cycle: synchronization and latching;
- 1 cycle: recognition;
- 2 cycles: branching to vector.
Total latency: 3 cycles.
Multiprocessor vector interrupts have 6 cycle latency.

**Supervisor mode**

- May want to provide protective barriers between programs.
- Avoid memory corruption.
- Need supervisor mode to manage the various programs.
- SHARC does not have a supervisor mode.

**ARM supervisor mode**

- Use SWI instruction to enter supervisor mode, similar to subroutine: `SWI CODE_1`
- Sets PC to 0x08.
- Argument to SWI is passed to supervisor mode code.
- Saves CPSR in SPSR.

**Exception**

- Exception: internally detected error.
- Exceptions are synchronous with instructions but unpredictable.
- Build exception mechanism on top of interrupt mechanism.
- Exceptions are usually prioritized and vectorized.

**Trap**

- Trap (software interrupt): an exception generated by an instruction.
  - Call supervisor mode.
- ARM uses SWI instruction for traps.
- SHARC offers three levels of software interrupts.
  - Called by setting bits in IRPTL register.
Co-processor

- **Co-processor**: added function unit that is called by instruction.
  - Floating-point units are often structured as co-processors.
  - ARM allows up to 16 designer-selected co-processors.
  - Floating-point co-processor uses units 1 and 2.

CPUs

- **Caches**.
- **Memory management**.

Caches and CPUs

- **CPU cache controller**.
- **Cache**.
- **Main memory**.

Cache operation

- Many main memory locations are mapped onto one cache entry.
- May have caches for:
  - instructions;
  - data;
  - data + instructions (unified).
- Memory access time is no longer deterministic.

Terms

- **Cache hit**: required location is in cache.
- **Cache miss**: required location is not in cache.
- **Working set**: set of locations used by program in a time interval.

Types of misses

- **Compulsory (cold)**: location has never been accessed.
- **Capacity**: working set is too large.
- **Conflict**: multiple locations in working set map to same cache entry.
Memory system performance

- \( h = \) cache hit rate.
- \( t_{\text{cache}} = \) cache access time, \( t_{\text{main}} = \) main memory access time.
- Average memory access time:
  \[
  t_{\text{av}} = ht_{\text{cache}} + (1-h)t_{\text{main}}
  \]

Multiple levels of cache

- \( h_1 = \) cache hit rate.
- \( h_2 = \) rate for miss on L1, hit on L2.
- Average memory access time:
  \[
  t_{\text{av}} = h_1t_{\text{L1}} + (1-h_1)h_2t_{\text{L2}} + (1-h_1)(1-h_2)t_{\text{main}}
  \]

Multi-level cache access time

- \( h_1 = \) cache hit rate.
- \( h_2 = \) rate for miss on L1, hit on L2.
- Average memory access time:
  \[
  t_{\text{av}} = h_1t_{\text{L1}} + (1-h_1)h_2t_{\text{L2}} + (1-h_1)(1-h_2)t_{\text{main}}
  \]

Replacement policies

- Replacement policy: strategy for choosing which cache entry to throw out to make room for a new memory location.
- Two popular strategies:
  - Random.
  - Least-recently used (LRU).

Cache organizations

- Fully-associative: any memory location can be stored anywhere in the cache (almost never implemented).
- Direct-mapped: each memory location maps onto exactly one cache entry.
- N-way set-associative: each memory location can go into one of n sets.

Cache performance benefits

- Keep frequently-accessed locations in fast cache.
- Cache retrieves more than one word at a time.
  - Sequential accesses are faster after first access.
Direct-mapped cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xabcd</td>
<td></td>
<td>byte</td>
</tr>
</tbody>
</table>

Write operations

- **Write-through**: immediately copy write to main memory.
- **Write-back**: write to main memory only when location is removed from cache.

Direct-mapped cache locations

- Many locations map onto the same cache block.
- Conflict misses are easy to generate:
  - Array a[] uses locations 0, 1, 2, ...
  - Array b[] uses locations 1024, 1025, 1026, ...
  - Operation a[i] + b[i] generates conflict misses.

Example: direct-mapped vs. set-associative

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0101</td>
</tr>
<tr>
<td>001</td>
<td>1111</td>
</tr>
<tr>
<td>010</td>
<td>0000</td>
</tr>
<tr>
<td>011</td>
<td>0110</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>101</td>
<td>0001</td>
</tr>
<tr>
<td>110</td>
<td>1010</td>
</tr>
<tr>
<td>111</td>
<td>0100</td>
</tr>
</tbody>
</table>

Set-associative cache

- A set of direct-mapped caches:
  - Set 1
  - Set 2
  - ... Set n

Example: direct-mapped cache behavior

<table>
<thead>
<tr>
<th>After 001 access:</th>
<th>After 010 access:</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>tag</td>
</tr>
<tr>
<td>00</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
</tr>
</tbody>
</table>
Direct-mapped cache behavior, cont’d.

- After 011 access:
  - Block: 00
  - Tag: -
  - Data: 00

- After 100 access:
  - Block: 00
  - Tag: 1
  - Data: 1000

- After 011 access:
  - Block: 00
  - Tag: 0
  - Data: 0111

- After 100 access:
  - Block: 00
  - Tag: 1
  - Data: 0000

2-way set-associative cache behavior

- Final state of cache (twice as big as direct-mapped):
  - Set: 0
  - Block: 0
  - Tag: 0
  - Data: 1000

- After 101 access:
  - Block: 00
  - Tag: 1
  - Data: 1000

- After 111 access:
  - Block: 00
  - Tag: 1
  - Data: 0001

2-way set-associative cache behavior

- Final state of cache (same size as direct-mapped):
  - Set: 0
  - Block: 0
  - Tag: 0
  - Data: 1000

- After 101 access:
  - Block: 00
  - Tag: 1
  - Data: 0001

- After 111 access:
  - Block: 00
  - Tag: 1
  - Data: 0000

Example caches

- StrongARM:
  - 16 Kbyte, 32-way, 32-byte block instruction cache.
  - 16 Kbyte, 32-way, 32-byte block data cache (write-back).

- SHARC:
  - 32-instruction, 2-way instruction cache.

Memory management units

- Memory management unit (MMU) translates addresses:
  - CPU
  - Logical address
  - Memory management unit
  - Physical address
  - Main memory
**Memory management tasks**

- Allows programs to move in physical memory during execution.
- Allows virtual memory:
  - Memory images kept in secondary storage;
  - Images returned to main memory on demand during execution.
- **Page fault**: request for location not resident in memory.

**Address translation**

- Requires some sort of register/table to allow arbitrary mappings of logical to physical addresses.
- Two basic schemes:
  - Segmented;
  - Paged.
- Segmentation and paging can be combined (x86).

**Segments and pages**

![Diagram of segments and pages]

**Segment address translation**

- **Segment base address**
- **Logical address**
- **Range check**
- **Physical address**
- **Range error**

**Page address translation**

- **Page base**
- **Page offset**
- **Concatenation**

**Page table organizations**

- **Flat**
- **Tree**
- **Page descriptor**
**Caching address translations**

- Large translation tables require main memory access.
- TLB: cache for address translation.
  - Typically small.

**ARM memory management**

- Memory region types:
  - section: 1 Mbyte block;
  - large page: 64 kbytes;
  - small page: 4 kbytes.
- An address is marked as section-mapped or page-mapped.
- Two-level translation scheme.

**ARM address translation**

Translation table base register

1st index 2nd index offset

concatenate

concatenate

physical address

**CPUs**

- CPU performance
- CPU power consumption.

**Elements of CPU performance**

- Cycle time.
- CPU pipeline.
- Memory system.

**Pipelining**

- Several instructions are executed simultaneously at different stages of completion.
- Various conditions can cause pipeline bubbles that reduce utilization:
  - branches;
  - memory system delays;
  - etc.
Pipeline structures

- Both ARM and SHARC have 3-stage pipes:
  - fetch instruction from memory;
  - decode opcode and operands;
  - execute.

Performance measures

- Latency: time it takes for an instruction to get through the pipeline.
- Throughput: number of instructions executed per time period.
- Pipelining increases throughput without reducing latency.

ARM pipeline execution

- Both ARM and SHARC have 3-stage pipes:
  - fetch instruction from memory;
  - decode opcode and operands;
  - execute.

Pipe stalls

- If every step cannot be completed in the same amount of time, pipeline stalls.
- Bubbles introduced by stall increase latency, reduce throughput.

Control stalls

- Branches often introduce stalls (branch penalty).
  - Stall time may depend on whether branch is taken.
- May have to squash instructions that already started executing.
- Don’t know what to fetch until condition is evaluated.
**ARM pipelined branch**

```
bne foo  
  fetch  
  decode
  ex      
  bne  
  ex  
  bne
```

**Delayed branch**

- To increase pipeline efficiency, delayed branch mechanism requires $n$ instructions after branch always executed whether branch is executed or not.
- SHARC supports delayed and non-delayed branches.
  - Specified by bit in branch instruction.
  - 2 instruction branch delay slot.

---

**Example: SHARC code scheduling**

```
L1=5;
DM(I0,M1)=R1;
L8=8;
DM(I8,M9)=R2;
```

- CPU cannot use DAG on cycle just after loading DAG's register.
- CPU performs NOP between register assign and DM.

---

**Rescheduled SHARC code**

```
L1=5;
L8=8;
DM(I0,M1)=R1;
DM(I8,M9)=R2;
```

- Avoids two NOP cycles.

---

**Example: ARM execution time**

- Determine execution time of FIR filter:
  ```
  for (i=0; i<N; i++)
    f = f + c[i]*x[i];
  ```
- Only branch in loop test may take more than one cycle.
  - BLT loop takes 1 cycle best case, 3 worst case.

---

**Superscalar execution**

- Superscalar processor can execute several instructions per cycle.
  - Uses multiple pipelined data paths.
- Programs execute faster, but it is harder to determine how much faster.
**Data dependencies**
- Execution time depends on operands, not just opcode.
- Superscalar CPU checks data dependencies dynamically:
  
  add r4, r0, r1
  add r3, r2, r5

**Memory system performance**
- Caches introduce indeterminacy in execution time.
  - Depends on order of execution.
- Cache miss penalty: added time due to a cache miss.
  - Several reasons for a miss: compulsory, conflict, capacity.

**CPU power consumption**
- Most modern CPUs are designed with power consumption in mind to some degree.
- Power vs. energy:
  - Heat depends on power consumption;
  - Battery life depends on energy consumption.

**CMOS power consumption**
- Voltage drops: power consumption proportional to $V^2$.
- Toggling: more activity means more power.
- Leakage: basic circuit characteristics; can be eliminated by disconnecting power.

**CPU power-saving strategies**
- Reduce power supply voltage.
- Run at lower clock frequency.
- Disable function units with control signals when not in use.
- Disconnect parts from power supply when not in use.

**Power management styles**
- Static power management: does not depend on CPU activity.
  - Example: user-activated power-down mode.
- Dynamic power management: based on CPU activity.
  - Example: disabling off function units.
**Application: PowerPC 603 energy features**

- Provides doze, nap, sleep modes.

**Dynamic power management features:**
- Uses static logic.
- Can shut down unused execution units.
- Cache organized into subarrays to minimize amount of active circuitry.

**PowerPC 603 activity**

- Percentage of time units are idle for SPEC integer/floating-point:
  
<table>
<thead>
<tr>
<th>unit</th>
<th>Specint92</th>
<th>Specfp92</th>
</tr>
</thead>
<tbody>
<tr>
<td>D cache</td>
<td>29%</td>
<td>28%</td>
</tr>
<tr>
<td>I cache</td>
<td>29%</td>
<td>17%</td>
</tr>
<tr>
<td>load/store</td>
<td>35%</td>
<td>17%</td>
</tr>
<tr>
<td>fixed-point</td>
<td>38%</td>
<td>76%</td>
</tr>
<tr>
<td>floating-point</td>
<td>99%</td>
<td>30%</td>
</tr>
<tr>
<td>system register</td>
<td>89%</td>
<td>97%</td>
</tr>
</tbody>
</table>

**Power-down costs**

- Going into a power-down mode costs:
  - time;
  - energy.
- Must determine if going into mode is worthwhile.
- Can model CPU power states with power state machine.

**Application: StrongARM SA-1100 power saving**

- Processor takes two supplies:
  - VDD is main 3.3V supply.
  - VDDX is 1.5V.
- Three power modes:
  - Run: normal operation.
  - Idle: stops CPU clock, with logic still powered.
  - Sleep: shuts off most of chip activity; 3 steps, each about 30 ms; wakeup takes > 10 ms.

**SA-1100 power state machine**

\[ P_{\text{run}} = 400 \text{ mW} \]
\[ P_{\text{idle}} = 50 \text{ mW} \]
\[ P_{\text{sleep}} = 0.16 \text{ mW} \]