Recall that ...

Steps in co-design
- Modeling
- Partitioning (into HW and SW parts)
- Scheduling
- Software Analysis (to support scheduling)
- Compilers + Processors to run the software
  - Similarly, synthesizing the hardware
- All of the above, with power management and not just performance in mind.

Organization

- Connection with the physical world
  - Sensors and Actuators
  - A/D and D/A converters
- Within the nice digitized world
  - ASICs
  - Processors
    - General-purpose Processors, Special-purpose processors (ASIP) and Custom processors.
    - Emphasis on Power management/Customization
  - Reconfigurable Logic

Connection to Environment

Sensors

Why need them?

Sense continuous variables from the physical world.
- Current, Voltage
- Temperature, Pressure, Light ...

Examples
- Proximity sensor in a car
  - Many modern cars have this particularly when the car is being driven in Reverse Gear
- Rain Sensor (in high-end cars)
  - Allow adjusting speed of wiper to rain speed.
- Image Sensors
  - Digital Camera, Fingerprint Authentication
Image Sensors based on CMOS

Smart Sensors

- Many applications
  - Target tracking/spying in difficult terrains.
  - Future Generation Home
    - Home 2015 --- Light turns on when you enter
  - Come under Distributed Sensor Networks
  - Slightly different from our presentation.
  - Our sensors are typically simplified.
  - They provide data to the processing units.
  - Integration of limited intelligence into sensors remains an active research area.

Smart Sensors

- Many applications
  - Target tracking/spying in difficult terrains.
  - Future Generation Home
    - Home 2015 --- Light turns on when you enter
  - Come under Distributed Sensor Networks
  - Slightly different from our presentation.
  - Our sensors are typically simplified.
  - They provide data to the processing units.
  - Integration of limited intelligence into sensors remains an active research area.

Actuators

- Provide output to the environment
  - Can vary radically in size and function
    - To be inserted into Human body or
    - Huge actuators overseeing mechanical movement of heavyweight items
  - Example from Health-care
    - Sensors --- Can be slapped/inserted into body to detect fail, monitor blood pressure etc.
    - Actuators --- Can be injected to enable release of drugs into the human body based on sensor data (and its processing)
      - Less intrusive than using injection syringe!

D/A Converters

- Convert digital data (d) to analog (a)
  - Can be fed into actuators
- Simple circuit
  - Often forms the basis of A/D Converters
- Basic Concept
  - Use each bit of (d) to control a current source
  - Arrange current sources in geometric progression depending on position of bits
  - Add current sources to produce total current at a junction
  - \( a = Output \ analog \ voltage \times Total \ Current \)

D/A Converter: Schematic

D/A Converter
A/D Converter
- Convert an analog voltage to a n-bit digital output
  - Given the maximum reference voltage $V_{ref}$
- Obvious cases
  - Voltage = $V_{ref}$ corresponds to 111...1
  - Voltage = 0 corresponds to 000...0
    - The minimum voltage can be different from 0 in which the definition of 000...0 can be suitably altered
- Use proportionality to convert the intermediate voltages to intermediate n-bit numbers
  - How to build a circuit to accomplish this task?

Voltage Comparison
- In general
  - $a / V_{ref} = d / 2^n$
  - $a$ = analog voltage value
  - $d$ = converted digital output for $a$
  - $V_{ref}$ = Maximum reference voltage
  - $n$ = precision of digital output (# of bits)
- Example
  - $d = (1.4/4)*4$
  - $= 1.4 = 1$ (rounded down)
    - Could be rounded up, then formula needs to change!
  - $= 01$ (digital output for 1.4V voltage)

Voltage Comparator Circuit
- Compares two analog voltages and produces a one-bit output
  - Out = 1 if $V_1 > V_2$
  - = 0 otherwise.

Flash A/D Comparator
- Say $n = 2$, $V_{ref} = 4V$, $a = 1.4V$
  - Find digital output $d$
  - Again define
    - $00 = 0-1V$, $01 = 1-2V$, $10 = 2-3V$, $11 = 3-4V$
  - Now, compare
    - $a > 4V$ (use voltage comparator)
      - No
    - $a > 3V$ (use another voltage comparator !)
      - No
  - Sequential search, and ...
    - Requires $2^n = 1$ comparators for n-bit output
**Flash A/D Converter**

- Sequential search parallelized in hardware — BUT
- Exponentially many voltage comparators required.

**Single Slope A/D converter**

- Can we reduce the number of voltage comparators?
- Say \( n = 2, V_{ref} = 4V, a = 1.4V \)
- Compute voltages for 11, 10, 01, 00 using a D/A converter.
- 4V, 3V, 2V, 1V
- Now compare
  - Analog input (1.4V) against voltage for 11 (4V)
  - 1.4 V < 4V, so compare 1.4V against voltage for 10
- We still perform sequential search
  - Maximum \( 2^n - 1 \) comparisons, all by same comparator
  - Very slow!!

**Example**

- \( n = 2, a = 1.4 \text{ V}, V_{ref} = 4 \text{ V} \)
  - \( \frac{1}{2} (4 + 0) = 2 \text{ V} > 1.4 \text{ V} \) 00
  - \( \frac{1}{2}(2 + 0) = 1 \text{ V} < 1.4 \text{ V} \) 01
  - Final encoding = 01

- Can you repeat the work for 4 bits?

**Successive Approx. A/D Converter**

- Connection with the physical world
  - Sensors and Actuators
  - A/D and D/A converters
- Within the nice digitized world
  - ASICs
  - Processors & Memory
    - General-purpose Processors, Special-purpose processors and Custom processors.
    - Emphasis on Power management/Customization
  - Reconfigurable Logic

**Organization**

- Connection with the physical world
  - Sensors and Actuators
  - A/D and D/A converters
- Within the nice digitized world
  - ASICs
  - Processors & Memory
    - General-purpose Processors, Special-purpose processors and Custom processors.
    - Emphasis on Power management/Customization
  - Reconfigurable Logic
### ASICs
- Implement functionality in custom hardware
  - Very efficient
  - No flexibility
  - Can only be done for very specific parts of the design.

### General-Purpose Processor
- Processor designed for a variety of computation tasks
  - Low unit cost, in part because manufacturer spreads NRE over large numbers of units
  - Carefully designed as high NRE is acceptable
    - Can yield good performance, size and power
  - Low cost, short time-to-market, high flexibility
    - User just writes software; no processor design

### Basic Architecture
- Control unit and data-path
- Features
  - General data-path
  - Control unit

### Data Path
- Register File
  - Storing intermediate results
- Arithmetic Logic Unit
  - Computations
- Operations may involve computation, movement across registers and/or mem.

### Data-path Operations
- Load
  - Read memory location into register
- ALU
  - Arithmetic/logical operation
- Store
  - Write register into memory location

### Control Unit
- Control unit: configures the data-path operations
  - Sequence of desired operations ("instructions") stored in memory - "program"
  - Instruction cycle – broken into several sub-operations, each one clock cycle:
    - Fetch: Get next instruction into IR
    - Decode: Determine what the instruction means
    - Fetch operands: Move data from memory to data-path register
    - Execute: Move data through the ALU
    - Store results: Write data from register to memory
Architectural Considerations

> **N-bit processor**
  - N-bit ALU, registers, buses, memory data interface
  - Embedded: 8-bit, 16-bit, 32-bit common
  - Desktop/servers: 32-bit, even 64-bit ALU, registers, buses, memory data interface

**Clock frequency**
- Inverse of clock period
- Must be longer than longest register to register delay in entire processor
- Memory access is often the longest

**Pipelining:** Increasing Instruction Throughput

**Superscalar and VLIW**
- Multiple ALU to support more than one instruction stream
  - Superscalar: Fetches instructions in batches, executes as many as possible
  - May require extensive hardware to detect independent instructions
  - VLIW: each word in memory has multiple independent instructions
    - Relies on the compiler to detect and schedule instructions
    - Currently growing in popularity --- many multimedia or DSP processors are VLIW processors.

**Very long instruction word (VLIW) processors**
Key idea: detection of possible parallelism to be done by compiler, not by hardware at run-time (inefficient).
VLIW: parallel operations (instructions) encoded in one long word (instruction packet), each instruction controlling one functional unit. E.g.:

<table>
<thead>
<tr>
<th>Instruction packet</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Instruction 3</th>
<th>Instruction 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Floating point unit</td>
<td>Integer unit</td>
<td>Integer unit</td>
<td>Memory unit</td>
</tr>
</tbody>
</table>

The Texas Instruments TMS 320C6xx as an example

- Bit in each instruction encodes end of parallel execution
- Instructions B, C and D cannot use any of the same functional units, cross paths or other data path resources. The same is also true for E, F and G.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B C D</td>
</tr>
<tr>
<td>3</td>
<td>E F G</td>
</tr>
</tbody>
</table>

Parallel execution cannot span several packets.
Partitioned register files

- Many memory ports are required to supply enough operands per cycle.
- Memories with many ports are expensive.
- Registers are partitioned into (typically 2) sets, e.g., for TI C60x:
  - Data path A
  - Data path B

The M3 VLIW DSP Processor

- The M3 VLIW DSP Processor is designed at TU Dresden (G. Fettweis et al.)
- It has 256 bit-wide memory and a large number of delay slots for processing instructions.
- Sample code:
  - if (a > 0) { x = 1; ... } y = x + z
    - In this case you cannot move instr in y = x + z to delay slots.
  - if (x > 0) { a = 1; b = 2; } y = x + z
    - You can move instructions in y = x + z to delay slots.
  - if (a > 0) { x = 1; y = 2 } else { x = 1; y = 0; }
    - You can move instructions in x = 1 to delay slots.
Predication

- Sometimes, it may be difficult to find instructions to fill delay slots.
- Another way is to get rid of branches and produce more and more straight line code
  - Predication!

Predicated execution

Conditional Instruction \([c]I\)
consists of:
- condition \(c\)
- instruction \(I\)

\[
\begin{align*}
  c = \text{true} & \implies I \text{ executed} \\
  c = \text{false} & \implies \text{NOP}
\end{align*}
\]

Execute/don’t execute individual instructions based on condition

\[
\begin{align*}
  \text{if} \ (a = 0) & \quad \text{pred}_{eq} \ p1, \ a, \ #0 \\
  b &= 4; & \quad \text{mov} \ b, \ #4 \ (p1)
\end{align*}
\]

Utility

Eliminate branch delay (also called if-conversion)
Predication makes delayed branches more effective - can fill delay slots with predicated instructions from both branch paths.

\[
\begin{align*}
  \text{if} \ (x = 0) & \quad bne \ #false, \ x, \ #0 \\
  y &= 2; & \quad \text{mov} \ y, \ #2 \\
  \text{else} & \quad \text{jmp} \ #\text{end} \\
  y &= 4; & \quad \text{false}: \ \text{mov} \ y, \ #4 \\
  \text{end}: & \quad \text{...}
\end{align*}
\]

What will be the compiled code with and without predicated execution?

Compiled code

- \(bne \ #false, \ x, \ #0\)
- \(\text{eq} \ p1, \ x, \ #0\)
- \(\text{mov} \ y, \ #2\)
- \((p1) \text{mov} \ y, \ #2\)
- \(\text{jmp} \ #\text{end}\)
- \((!p1) \text{mov} \ y, \ #4\)
- \(\text{false}: \ \text{mov} \ y, \ #4\)
- \(\text{end}: \ ...\)

No branches in this one!!

Predication removes branches and forms larger chunks of straight-line code often called hyperblocks in compiler literature.
Predication involves changes to instruction set, however minor.
Increases instruction fetch, predicated instructions are always fetched.

General Processors - Recap

- Instruction scheduling
  - At run-time --- Superscalar
  - At compile-time --- VLIW
    - Important for the embedded domain (Why?)
- Instruction set Architecture
  - Reduced --- RISC machines
  - Complex --- CISC machines
- Much processing in embedded systems is still done by micro-controllers
  - Simple micro-architecture, instruction set and little on-chip memory.

Application-Specific Instruction-Set Processors (ASIP)

- General-purpose processors
  - Sometimes too general to be effective in demanding application
    - e.g., video processing – requires huge video buffers and operations on large arrays of data, inefficient on a GPP
  - But ASIC has high NRE, not programmable
- ASIP – targeted to a particular domain
  - Contain architectural features specific to that domain
    - e.g., embedded control, digital signal processing, video processing, network processing, telecommunications, etc.
  - Still programmable
A Common ASIP: Digital Signal Processors (DSP)

- For signal processing applications
  - Large amounts of digitized data, often streaming
  - Data transformations must be applied fast
  - E.g., cell-phone voice filter, digital TV, music synthesizer
- DSP features
  - Does a lot of math operations
  - Several instruction execution units
  - Multiply-accumulate single-cycle instruction
  - Efficient vector operations, e.g., add two arrays
    - Vector ALU, loop buffers, etc.

Domain of DSP

- DSP process data in real time
- Analog information is converted to a digital representation, processed, then reconverted to analog

More on DSP

- The Killer App for DSP (16-bit fixed point) has been the PC modem and sound cards
  - In 1998, 70 to 80 Million PC are sold
  - Every modem and high-end sound card has a DSP
- New Killer App is predicted to be voice encoding/decoding
  - Voice Over Internet Protocol (VOIP)
  - Speech recognition

Features of DSP processors

- Support for Vector operations
- Special addressing modes
- Saturating and fixed point arithmetic
  - As opposed to floating point
  - Multiple memory banks

DSP Processors: new instr.

```
for (j=1 to n)
  [MR:=MR+MX*MY; MY:=a[A1]; MX:=x[A2]; A1++; A2--]
```

Multiply/accumulate (MAC) instruction

Zero-overhead loop (ZOL) instruction preceding MAC instruction.
Loop testing done in parallel to MAC operations.

Saturating arithmetic

- Returns largest/smallest number in case of over/underflows
- Example:
  - `a` = 0111
  - `b` = 1001
  - Standard wrap around arithmetic: `(`0000`)
  - Saturating arithmetic: 1111
  - `(a+b)/2`:
    - Correct: 1000
    - Wrap around arithmetic: 0000
    - Saturating arithmetic + shifted: 0111 (almost correct)
- Appropriate for DSP/multimedia applications:
  - No timeliness of results if interrupts are generated for overflows
  - Precise values less important
  - Wrap around arithmetic would be worse.
**Fixed-point arithmetic**

Shifting required after multiplications and divisions in order to maintain binary point.

**Properties of fixed-point arithmetic**

- Automatic scaling a key advantage for multiplications.
- Example:
  
  \[ x = 0.5 \times 0.125 + 0.25 \times 0.125 = 0.0625 + 0.03125 = 0.09375 \]
  
  For \( iwl = 1 \) and \( fwl = 3 \) decimal digits, the less significant digits are automatically chopped off: \( x = 0.093 \)

  Like a floating point system with numbers \( c \in [0..1) \), with no stored exponent (bits used to increase precision).

- Appropriate for DSP/multimedia applications (well-known value ranges).

**Real-time capability**

- **Timing behavior has to be predictable**
  
  Features that cause problems:
  - Unpredictable access to shared resources
  - Caches with difficult to predict replacement strategies
  - Unified caches (conflicts betw. instructions and data)
  - Pipelines with difficult to predict stall cycles ("bubbles")
  - Unpredictable communication times for multiprocessors
  - Branch prediction, speculative execution
  - Interrupts that are possible any time
  - Memory refreshes that are possible any time
  - Instructions that have data-dependent execution times

  Trying to avoid as many of these as possible.

**e.g. Cache->scratchpad**

- One way to avoid the timing unpredictability due to cache
  - Compiler controlled scratchpad memory
  - At compile time define the contents of what is going inside cache
  - Cache contents are locked, do not change.
  - Possible to extend this with dynamic overlays -- cache contents changes at selected points.

  In more details in next lecture.

**Another Trend: Customized ASIPs**

- In the past, microprocessors acquired as chips
- Today, we increasingly acquire a processor as Intellectual Property (IP)
  - e.g., synthesizable VHDL model
- Opportunity to add a custom datapath hardware and a few custom instructions, or delete a few instructions
  - Can have significant performance, power and size impacts
  - Problem: need compiler/debugger for customized ASIP
    - Remember, most development uses structured languages
    - One solution: automatic compiler/debugger generation
      - e.g., www.tensilica.com
    - Another solution: retargetable compilers
      - e.g., www.improsys.com (customized VLIW architectures)

**Organization**

- Connection with the physical world
  - Sensors and Actuators
  - A/D and D/A converters
- Within the nice digitized world
  - ASICs
  - Processors & Memory
    - General-purpose Processors, Special-purpose processors and Custom processors.
  - Reconfigurable Logic
**So, what is it**

- **Programmable Logic Devices**
  - An integrated circuit chip that can be configured by end user to implement different digital hardware
  - Also known as "Field Programmable Logic Device (FPLD)"

**Reconfigurable Logic**

- Full custom chips may be too expensive, software too slow.
- Combine the speed of HW with flexibility of SW
  - HW with programmable functions and interconnect.
  - Use of configurable hardware;
    - common form: field programmable gate arrays (FPGAs)
  - Applications: bit-oriented algorithms like
    - encryption,
    - fast object recognition (medical and military)
    - Adapting mobile phones to different standards.
- Very popular devices from
  - XILINX (XILINX Vertex II are very recent devices)
  - Actel and others

**Exercise**

- Given a trace of instructions, perform opcode assignment to minimize number of bit-flips. # of bits in each opcode is fixed.
  - This may be common in embedded applications and processors
  - Traces obtained by profiling application
    - Often programs with one path.
  - Opcode assignment not fixed being designed.
  - Number of bit-flips indicative of energy consumption which may be an issue.

**Answer**

- Flip in a single bit position defined as
  - $| x[i] - y[i] |$ where $x[i]$, $y[i]$ are bits.
- This can be encoded in ILP as
  - $-c \leq x[i] - y[i] \leq c, \quad 0 \leq c \leq 1$
  - That is, $c$ is an extra ILP variable.
- Instead of minimizing $| x[i] - y[i] |$, now minimize $c$

**Assignment 3**

- Use Chronos WCET estimation tool for cache-aware scheduling
  - Due on Friday November 16.
  - Use the server modelchk.comp.nus.edu.sg
  - Discussion on assignment now.