Today we’ll take a look at Intel’s 8086, which is one of the oldest and yet most prevalent processor architectures around.

We’ll make many comparisons between the MIPS and 8086 architectures, focusing on registers, instruction operands, memory and addressing modes, branches, function calls and instruction formats.

This will be a good chance to review the MIPS architecture as well.
An x86 processor timeline

1971: Intel’s 4004 was the first microprocessor—a 4-bit CPU (like the one from CS231) that fit all on one chip.

1978: The 8086 was one of the earliest 16-bit processors.

1981: IBM uses the 8088 in their little PC project.

1989: The 80486 includes a floating-point unit in the same chip as the main processor, and uses RISC-based implementation ideas like pipelining for greatly increased performance.

1997: The Pentium II is superscalar, supports multiprocessing, and includes special instructions for multimedia applications.

2002: The Pentium 4 runs at insane clock rates (3.06 GHz), implements extended multimedia instructions and has a large on-chip cache.
MIPS registers

- The MIPS architecture supports 32 registers, each 32-bits wide.
- Some registers are reserved by convention.
  - `$zero` always contains a constant 0.
  - `$at` is used by assemblers in converting pseudo-instructions.
  - `$a0-$a3` store arguments for function calls.
  - `$v0-$v1` contain return values from functions.
  - `$ra` is the return address in function calls.
  - `$sp` is the stack pointer.
- There are some other reserved registers we didn’t mention:
  - `$k0-$k1`  
  - `$fp`
  - `$gp`
- Only registers `$t0-$t9` and `$s0-$s7` are really “free.”
8086 registers

- There are four general-purpose 32-bit registers.
  
  EAX    EBX    ECX    EDX

- Four other 32-bit registers are usually used to address memory.
  
  ESP    EBP    ESI    EDI

- Several 16-bit registers are used for the segmented memory model.
  
  CS     SS     DS     ES     FS     GS

- Finally, there are two special 32-bit registers:
  - EIP is the instruction pointer, or program counter.
  - EFLAGS contains condition codes for branch instructions.

- Having a limited number of general-purpose registers typically means that more data must be stored in memory, and more memory accesses will be needed.
MIPS instruction set architecture

- MIPS uses a **three-address, register-to-register** architecture

  ![Diagram](image)

  - This is interpreted as $a = b + c$.
  - $a$ and $b$ must be registers.
  - $c$ may be a register or, in some cases, a constant.
The 8086 is a two-address, register-to-memory architecture.

- This is interpreted as \( a = a + b \).
  - \( a \) can be a register or a memory address.
  - \( b \) can be a register, a memory reference, or a constant.
  - But \( a \) and \( b \) cannot both be memory addresses.
- There are also some one-address instructions, which leave the destination and first source implicit.
MIPS memory

- Memory is byte-addressable—each address stores an 8-bit value.
- Addresses can be up to 32 bits long, resulting in up to 4 GB of memory.
- The only addressing mode available is indexed addressing.

\[ \text{lw } \textit{t0}, \textit{20}($a0) \quad \# \textit{t0} = M[$a0 + 20] \]
\[ \text{sw } \textit{t0}, \textit{20}($a0) \quad \# M[$a0 + 20] = \textit{t0} \]

- The lw/sw instructions access one word, or 32 bits of data, at a time.
  - Words are stored as four contiguous bytes in memory.
  - Words must be aligned, starting at addresses divisible by four.
8086 memory

- Memory is also byte-addressable.
  - The original 8086 had a 20-bit address bus that could address just 1MB of main memory.
  - Newer CPUs can access 64GB of main memory, using 36-bit addresses.
- Since the 8086 was a 16-bit processor, some terms are different.
  - A word in the 8086 world is 16 bits, not 32 bits.
  - A 32-bit quantity is called a double word instead.
- Data does not have to be aligned. Programs can easily access data at any memory address, although performance may be worse.
A note on memory errors

- Modern operating systems prevent user programs from accessing memory that doesn’t belong to them.
- For instance, a segmentation fault or general protection fault occurs if a program tries to read from address 0—in other words, if dereferences a NULL pointer.
- A bus error happens when programs try to access non-aligned data, such as reading a word from location 0x400021 on the CSIL machines.

```c
int *p1 = (int *) 0x00000000;
int *p2 = (int *) 0x00400021;
x = *p1;
x = *p2;
```

- Intel 8086 processors and PCs don’t have this alignment restriction, which can create confusion when trying to port or debug programs.
Segments

- In the original 8086 registers are only 16-bits wide, and two registers are needed to produce a 20-bit memory address.
  - A segment register specifies the upper 16 bits of the address.
  - Another register specifies the lower 16 bits of the address.
- These registers are then added together in a special way.

```
4 bits
  \[\begin{array}{cccc}
    \hline
    & & & \\
  \end{array}\] \quad 16-bit segment register
+ \quad \[\begin{array}{cccc}
    & & & \\
  \end{array}\] \quad 16-bit offset register
\hline
= \quad \[\begin{array}{cccc}
    & & & \\
  \end{array}\] \quad 20-bit address
```

- A single 20-bit address can be specified in multiple ways! For instance, 0000:0040 is the same as 0004:0000 (in hexadecimal notation).
Segment examples

- Segments come into play in many situations.
  - The program counter is a 20-bit value CS:IP (the instruction pointer, within the code segment).
  - The stack pointer is really SS:SP.
- Many instructions use a segment register implicitly, so the programmer only needs to specify the second, offset register.
- Segments make programming more interesting.
  - Working with memory in one segment is simple, since you can just set a segment register once and then leave it alone.
  - But large data structures or programs that span multiple segments can cause a lot of headaches.
- The newer 8086 processors support a flat 32-bit address space in addition to this segmented architecture.
8086 addressing modes

- Immediate mode is similar to MIPS.
  \[
  \text{mov \ eax, 4000000} \quad # \text{eax} = 4000000
  \]

- Displacement mode accesses a given constant address.
  \[
  \text{mov \ eax, [4000000]} \quad # \text{eax} = M[4000000]
  \]

- Register indirect mode uses the address in a register.
  \[
  \text{mov \ eax, [ebp]} \quad # \text{eax} = M[ebp]
  \]

- Indexed addressing is similar to MIPS.
  \[
  \text{mov \ eax, [ebp+40]} \quad # \text{eax} = M[ebp+40]
  \]

- Scaled indexed addressing does multiplication for you.
  \[
  \text{mov \ eax, [ebx+esi*4]} \quad # \text{eax} = M[ebx+esi*4]
  \]

- You can add extra displacements (constants) and go crazy.
  \[
  \text{mov \ eax, 20[ebx+esi*4+40]} \quad # \text{eax} = M[ebx+esi*4+60]
  \]
Array accesses with the 8086

- **Scaled addressing** is valuable for stepping through arrays with multi-byte elements.
- In MIPS, to access word $t1$ of an array at $t0$ takes several steps.

  ```
  mul $t2, $t1, 4 # $t2 is byte offset of element $t1
  add $t2, $t2, $t0 # Now $t2 is address of element $t1
  lw $a0, 0($t2) # $a0 contains the element
  ```

- In 8086 assembly, accessing double word esi of an array at ebx is shorter.

  ```
  mov eax, [ebx+esi*4] # eax gets element esi
  ```

- You don’t have to worry about incrementing pointers by 4 or doing extra multiplications explicitly again!
MIPS branches and jumps

- MIPS has four basic instructions for branching and jumping.
  
  bne    beq    j    jr

- Other kinds of branches are split into two separate instructions.
  
  ```
  slt $at, $a0, $a1  # $at = 1 if $a0 < $a1
  bne $at, $0, Label # Branch if $at != 0
  ```

- `slt` uses a temporary register to store a Boolean value that is then tested by a `bne/beq` instruction.

- Together, branches and jumps can implement conditional statements, loops, and function returns.
8086 branches and jumps

- The 8086 chips contain a special register of status flags, EFLAGS.
- The bits in EFLAGS are adjusted as a side effect of arithmetic and special test instructions.
- Some of the flags, which might look familiar from CS231, are:
  - $S = 1$ if the ALU result is negative.
  - $O = 1$ if the operation caused a signed overflow.
  - $Z = 1$ if the result was zero.
  - $C = 1$ if the operation resulted in a carry out.
- The 8086 ISA provides instructions to branch (they call them jumps) if any of these flags are set or not set.

  js/jns     jo/jno     jz/jnz     jc/jnc
MIPS function calls

- The `jal` instruction saves the address of the next instruction in `$ra` before transferring control to a function.
- Conventions are used for passing arguments (in `$a0-$a3`), returning values (in `$v0-$v1`), and preserving caller-saved and callee-saved registers.
- The stack is a special area of memory used to support functions.
  - Functions can allocate a private stack frame for local variables and register preservation.
  - Stack manipulations are done explicitly, by modifying `$sp` and using load/store instructions with `$sp` as the base register.
8086 function calls

- Control flow for 8086 function calls involves two aspects.
  - The `CALL` instruction is similar to `jal` in MIPS, but the return address is placed on the `stack` instead of in a register.
  - `RET` pops the return address on the stack and jumps to it.

- The flow of data in 8086 function calls faces similar issues as MIPS.
  - Arguments and return values can be passed either in registers or on the stack.
  - Functions are expected to preserve the original values of any registers they modify—in other words, all registers are callee-saved.

- The 8086 also relies upon a stack for local storage.
  - The stack can be manipulated explicitly, via the `esp` register.
  - The CPU also includes special `PUSH` and `POP` instructions, which can manage the stack pointer automatically.
MIPS instruction formats

- There are just three MIPS instruction formats: R-type, I-type and J-type.
- The formats are very uniform, leading to simpler hardware.
  - Each instruction is the same length, 32 bits, so it’s easy to compute instruction addresses for branch and jump targets.
  - Fields are located in the same relative positions when possible.
- These formats are sufficient to encode most operations. Less common operations are implemented with multiple MIPS instructions.
8086 instruction formats

- Instruction formats range wildly in size from 1 to 17 bytes, mostly due to all the complex addressing modes supported.
- This means more work for both the hardware and the assembler.
  - Instruction decoding is very complex.
  - It’s harder to compute the address of an arbitrary instruction.
- Things are also confusing for programmers.
  - Some instructions appear in two formats—a simpler but shorter one, and a more general but longer one.
  - Some instructions can be encoded in different but equivalent ways.
CISC

- When the 8086 was introduced, memory was hideously expensive and not especially fast.
- Keeping the encodings of common instructions short helped in two ways.
  - It made programs shorter, saving precious memory space.
  - Shorter instructions can also be fetched faster.
- But more complex, longer instructions were still available when needed.
  - Assembly programmers often favored more powerful instructions, which made their work easier.
  - Compilers had to balance compilation and execution speed.
- The 8086-based processors are an example of a complex instruction set computer, or CISC, architecture.
Many newer processor designs use a reduced instruction set computer, or RISC, architecture instead.

The idea of simpler instructions and formats seemed radical in the 1980s.

- RISC-based programs needed more instructions and were harder to write by hand than CISC-based ones.
- This also meant that RISC programs used more memory.

But this has obviously worked out pretty well.

- Memory is faster and cheaper now.
- Compilers generate code instead of assembly programmers.
- Simpler hardware made advanced implementation techniques like pipelining easier and more practical.
Current Intel CPUs

- It wasn’t until the Pentiums that Intel really started “leading.”
  - They used to have inferior floating-point units.
  - Requiring compatibility with the old, complex 8086 made it hard to implement pipelining and superscalar architectures until recently.
  - Overall performance suffered.

- The Pentiums now use many RISC-based implementation ideas.
  - All complex 8086 instructions are translated into sequences of simpler “RISC core” instructions.
  - This makes pipelining possible—in fact, the Pentium 4 has the deepest pipeline in the known universe, which helps it run up to 3 GHz.
  - Modern compilers and programmers avoid the slower, inefficient instructions in the ISA, which are provided only for compatibility.

- New Pentiums also include additional MMX, SSE and SSE2 instructions for parallel computations, which are common in image and audio processing applications.
A word about cheapness

- The original IBM PC used the 8088, which was an 8086 with an 8-bit data bus instead of a 16-bit one.
  - This made it cheaper to design, and it could maintain compatibility with existing 8-bit memories, chipsets and other hardware.
  - The registers were still 16 bits, so two cycles were needed to transfer data between a register and memory.
- Intel pulled the same trick in the late 80s, with the 80386SX and its 16-bit data bus, compared to the regular 80386’s 32-bit bus.
- Today there are still “value” CPUs like Intel’s Celeron and AMD’s Duron, which have smaller caches and/or slower buses than their more expensive counterparts.
Ways to judge CPUs

- Computer systems always try to balance price and performance. Cheaper processors often—but not always—have lower performance.
- When power consumption is important, Intel offers Mobile Pentiums and AMD has Mobile Athlons. There are also many other low-power processors including the Transmeta Crusoe and IBM/Motorola PowerPC.
- Intel is still expanding the 8086 instruction set, with the newer MMX, SSE, and SSE2 instructions.
- The Pentium’s compatibility with older processors is a strength, but also weakness that may impede enhancements to the CPU design.
  - Intel is designing the Itanium, a new 64-bit processor, from scratch.
  - In contrast, AMD is making a 64-bit, backward compatible extension to the 8086 architecture.
Pentium 4 pictures

Top

Bottom

Pictures are taken from http://www.anandtech.com
An older AMD Athlon

More pictures from http://www.anandtech.com
Summary

- The MIPS architecture we’ve seen so far is fairly simple, especially when compared to the Intel 8086 series.
- The basic ideas in most processors are similar, though.
  - Several general-purpose registers are used.
  - Simple branch and jump instructions are needed for control flow.
  - Stacks and special instructions implement function calls.
  - A RISC-style core leads to simpler, faster hardware.