CS61C – Machine Structures

Lecture 40 - X86 Processors

5/1/2006

John Wawrzynek

(www.cs.berkeley.edu/~johnw)

www-inst.eecs.berkeley.edu/~cs61c/

Outline

° History of Intel x86 line.

° MIPS versus x86

° Unusual features of x86

° Internal details of implementations

X86

From Wikipedia, the free encyclopedia.

- x86 or Intel 80x86 is the generic name of an architecture of microprocessors first developed and manufactured by Intel, also manufactured at various stages by AMD, Cyrix, NEC, Transmeta (that uses it in PDAs too, see Crusoe) and sundry other makers at various stages in its nearly 25-year history.
- In addition to basic architecture itself, these names are also used to describe a family of particular microprocessors manufactured by Intel, including the Intel 8086, Intel 80186, Intel 80286, Intel 80386, Intel 80486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Pentium M,. The architecture of Intel's 32-bit x86 processors is sometimes known as IA-32.
- Intel's IA-64 architecture used in its Itanium processors is related to x86, but incompatible with its instruction set. AMD's x86-64 is backward-compatible with x86.

Intel History: ISA evolved since 1978

- 8086: 16-bit, all internal registers 16 bits wide; no general purpose registers; ’78
- 8087: + 60 Fl. Pt. instructions, (Prof. Kahan) adds 80-bit-wide stack, but no general purpose registers; ‘80
- 8088: simpler version of 8086 adopted as standard CPU of the IBM PC; ‘81
- 80286: expands addressable memory to 16MB (from 1MB), adds elaborate protection model; ‘82
- 80386: 32-bit; converts 8 16-bit registers into 8 32-bit general purpose registers; new addressing modes; adds paging to support OS; ‘85
- 80486, Pentium, Pentium Pro: + 4 instructions
- MMX: + 57 instructions for multimedia; ’96
- Pentium III: +70 instructions for multimedia; ‘99
- Pentium 4: +144 instructions for multimedia; '00
- (AMD extends ISA to 64 bits; ’02)

x86 design decisions

° Early x86 were designed to be hand programmed as much as compiled to
  - Thus, complicated instructions (e.g., string copy) which make a programmer's life easier were good
    - Current x86 processors will translate these on the fly

° Memory was very expensive
  - So keeping code size small was very important

° Registers were very expensive

° Backwards compatibility is king!
  - Thus can only add to the ISA, never take away

MIPS versus 80386

MIPS	80386
° Address: 32-bit	° 32-bit
° Page size: 4KB	° 4KB
° Data aligned	° Data unaligned
° Destination reg: Left	° Right
• `add $rd,$rs1,$rs2`	• `add %rs1,%rs2,%rd`
° Regs: $0, $1, ..., $31	° %r0, %r1, ..., %r7
° Reg = 0: $0	° (n.a.)
° Return address: $31	° return address is automatically saved on the stack

MIPS versus Intel 80x86

° MIPS: "Three-address architecture"

- Arithmetic-logic specify all 3 operands

```
add $s0,$s1,$s2 # s0=s1+s2
```

- Benefit: fewer instructions ⇒ performance

° x86: "Two-address architecture"

- Only 2 operands,
  so the destination is also one of the sources

```
add $s1,$s0 # s0=s0+s1
```

- Often true in C statements: `c += b;`
  - Also present in 70s era micro-architectures, such as the early VAXes, which is why C has such operators
    - C is portable VAX assembly
- Benefit: smaller instructions ⇒ smaller code
  - Code size was MUCH more important when the x86 was first designed
- Cost: May require more register moves

MIPS versus Intel 80x86

° MIPS: "load-store architecture"

- Only Load/Store access memory; rest operations register-register; e.g.,

```
lw $t0, 12($gp)
add $s0,$s0,$t0 # s0=s0+Mem[12+gp]
```

- Benefit: simpler hardware ⇒ easier to pipeline, higher performance
  - Only works well when one has plenty of registers

° x86: "register-memory architecture"

- All operations can have an operand in memory; other operand is a register; e.g.,

```
add 12(%gp),%s0 # s0=s0+Mem[12+gp]
```

- Benefit: fewer instructions in the program ⇒ smaller code
- Cost: More complicated hardware, more instructions to implement

MIPS versus Intel 80x86

- MIPS: "fixed-length instructions"
  - All instructions same size, e.g., 4 bytes
  - simple hardware ⇒ performance
  - branches can be multiples of 4 bytes
- x86: "variable-length instructions"
  - Instructions are multiple of bytes: 1 to 17;
    - Simple, common instructions should be smaller
    - small code size (30% smaller?)
      - Provides for a better icache hit rate, if the icache stores instructions
    - But significantly complicates decoding
  - Instructions can include 8- or 32-bit immediates

MIPS versus Intel 80x86

- MIPS: "fixed-length operations"
  - All operations on same data size: 4 bytes; whole register changes
  - Goal: simple hardware and high performance
- x86: "variable-length operations"
  - Operations are multiple of bytes: 1, 2, 4
  - Only part of register changes if op < 4 bytes
  - Condition codes are set based on width of operation for Carry, Sign, Zero
- X86: 16-bits called word; 32-bits double word or long word (halfword and word in MIPS)

MIPS is example of RISC

- RISC = Reduced Instruction Set Computer
  - Term coined at Berkeley, ideas pioneered by IBM, Berkeley, Stanford
- RISC characteristics:
  - Load-store architecture
  - Fixed-length instructions (typically 32 bits)
  - Three-address architecture
  - Plentiful registers
  - All instructions effectively take identical time
  - Designed for high performance operation and as a compiler target
- RISC examples: MIPS, SPARC, IBM/Motorola PowerPC, Compaq Alpha, ARM, SH4, HP-PA.

x86 is the classic CISC architecture

- CISC = Complex Instruction Set Computer
- General characteristics:
  - Instructions have greater operand types (constants, registers, memory)
  - Variable length instructions
    - Instruction latency may vary heavily between different instructions
  - Usually sparse registers
  - Designed to save code space and as a target for hand-written assembly
- x86 family, Motorola 68k series (pre PowerPC Macintoshes, Palm Pilot)

RISC versus CISC

CISC ⇒ more expensive implementation, lower-performance.

Why is it that the x86, being a CISC, has been as successful as it has?

1. **Business alliance with IBM – x86 became the standard processor for PCs.**
2. **IC manufacturing – Intel leads the world (with IBM) in state-of-the-art fabrication.**
3. **Newer x86 implementations adopt RISC features:**
   - **New RISC-like instructions**
   - **On-the-fly translation of complex instructions.**

Unusual features of 80x86

° **8 32-bit Registers have names; 16-bit 8086 names with "e" prefix:**
- **`eax`, `ecx`, `edx`, `ebx`, `esp`, `ebp`, `esi`, `edi`**
- **80x86 word is 16 bits, double word is 32 bits**

° **PC is called `eip` (instruction pointer)**

° **`leal` (load effective address) instruction**
- **Calculate address like a load, but load address into register, not data**
- **Load 32-bit address:**

```
leal -4000000(%ebp),%esi
 # esi = ebp - 4000000
```

Instructions:MIPS vs. 80x86

° `addu, addiu`	° `addl`
° `subu`	° `subl`
° `and,or, xor`	° `andl, orl, xorl`
° `sll, srl, sra`	° `sall, shrl, sarl`
° `lw`	° `movl mem, reg`
° `sw`	° `movl reg, mem`
° `mov`	° `movl reg, reg`
° `li`	° `movl imm, reg`
° `lui`	° `Not needed`

80386 addressing (ALU instructions too)

° **base reg + offset (like MIPS)**

```
•movl -8000044(%ebp), %eax
```

° **base reg + index reg (2 regs form addr.)**

```
•movl (%eax,%ebx),%edi
  # edi = Mem[ebx  + eax]
```

° **scaled reg + index (shift one reg by 1,2)**

```
•movl(%eax,%edx,4),%ebx
  # ebx = Mem[edx*4 + eax]
```

° **scaled reg + index + offset**

```
•movl 12(%eax,%edx,4),%ebx
  # ebx = Mem[edx*4 + eax + 12]
```

Branch in 80x86

- Rather than compare registers, x86 uses special 1-bit registers called “condition codes” that are set as a side-effect of ALU operations
  - S - Sign Bit
  - Z - Zero (result is all 0)
  - C - Carry Out
  - P - Parity: set to 1 if even number of ones in rightmost 8 bits of operation
- Conditional Branch instructions then use condition flags for all comparisons: <, <=, >, >=, ==, !=
  - Conditional execution and condition codes are also present in some RISC architectures, such as the ARM ISA

Branch: MIPS vs. 80x86

`beq`	`(cmpl;) je` if previous operation set condition code, then `cmpl` unnecessary
`bne`	`(cmpl;) jne`
`slt; beq`	`(cmpl;) jlt`
`slt; bne`	`(cmpl;) jge`
`jal`	`call`
`jr $31`	`ret`

While in C/Assembly: 80x86

C
```
    while (save[i]==k)
        i = i + j;
```

(i,j,k: `%edx,%esi,%ebx`)

x86
```
         leal -400(%ebp),%eax
.Loop:   cmpl %ebx,(%eax,%edx,4)
         jne .Exit
         addl %esi,%edx
         j .Loop
.Exit:
```

Note: cmpl replaces sll, add, lw in loop

Unusual features of 80x86

° Memory Stack is part of instruction set

- `call` places return address onto stack, increments `esp` (Mem[`esp`]=`eip`+6; `esp`+=4), as well as changing the flow of control
- `push` places value onto stack, increments `esp`
- `pop` gets value from stack, decrements `esp`

° `incl, decl` (increment, decrement)

```
incl %edx # edx = edx + 1
```

Unusual features of 80x86: Floating Pt.

° Floating point uses a separate stack; load, push operands, perform operation, pop result

```
 fildl (%esp)
    # fpstack = M[esp],
    # convert integer to FP
flds -8000048(%ebp)
    # push M[ebp-8000048]
fsubp %st,%st(1)
    # subtract top 2 elements
fstps -8000048(%ebp)
    # M[ebp-8000048] = difference
```

Conclusion

° Once you've learned one RISC instruction set, easy to pick up the rest
  - ARM, Compaq/DEC Alpha, Hitatchi SuperH, IBM/Motorola PowerPC, Sun SPARC, ...

° Intel 80x86 is a "horse of different color"
  - But still reasonably straightforward, albeit more complicated

° RISC emphasis: performance, HW simplicity, compiler targets

° 80x86 emphasis: code size and hand coding
  - A decision which was right at the time, but is no longer really relevant

Announcements:

- Midterm 3:
  - Tuesday 5/9, 6:30-9:30, 1 Pimentel
  - Format: 45 points (MT1&2 were 30 each)
    - 25 points on final 1/3 of semester (pipelining included)
    - 20 points on 61c “greatest hits”

      Basic C programming concepts,

      MIPS programming,

      Processor Design (datapath/control),

      Synchronous Digital System Design (data movement/Flip-flops, simple CL blocks).
  - TA lead review session, 2-4pm Sunday (5/7), 10 Evans.

Announcements:

- This week schedule:
  - Today: “x86” & Course Evaluation

    fill out course evaluation - VERY IMPORTANT

    Bring a #2 pencil.
  - Friday: Alternatives to Processors: FPGAs
- Next Week:
  - Monday: course wrap up, review
  - Tuesday evening exam
  - Grades out end of next week.

Extra Slides: micro-architecture and implementation details

Intel Internals

- Hardware below instruction set called "microarchitecture"
- Pentium Pro, Pentium II, Celeron, Pentium III are all based on same microarchitecture (1994)
    - Improved clock rate (from process shrinks), increased cache size, some minor design tweaks
- AMD Athlon/Duron is a different beast
- Pentium 4 had new microarchitecture
- Pentium M based on Pentium 3

Dynamic Scheduling in Pentium Pro, II, III

- PPro doesn't pipeline 80x86 instructions
  - Instead, it translates each x86 instruction into one or more 72 bit "micro-operations" (μOps)
- It takes 1 clock cycle to determine the length of the x86 instruction + 2 more to create the μOps
  - Most instructions translate into 1-4 μOps
  - The PPro can translate up to 3 instructions into 5 μOps/cycle if the instructions are ordered correctly
- 10 stage pipeline for micro-operations

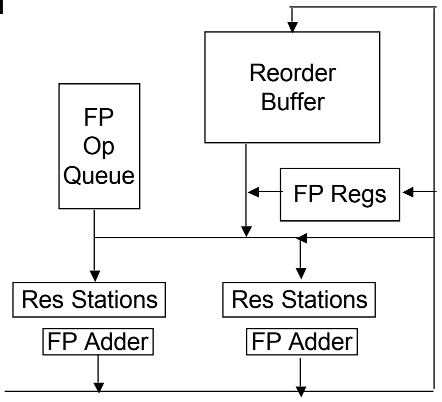
Hardware support

- *Out-of-Order execution*: allow a instructions to execute in a different order than they appear in the instruction stream.
  - An instruction waits in a queue until all of its input operands are ready, then gets executes on the appropriate "function unit"
- Fetch in-order, execute out-of-order, commit (change processor state) in order
  - Necessary for precise exceptions if something goes wrong
    - It is necessary to insure that every instruction after the bad one never executed

Hardware for out of order execution

° Need HW buffer for results of uncommitted instructions: *reorder buffer*

- Reorder buffer can be operand source
- Once operand "commits", result is found in register
- Discard results on mispredicted branches or on exceptions



Dynamic Scheduling in Pentium Pro

Max. instructions issued/clock	3
Max. μOps issued/clock	5
Max. μOps complete exec./clock	5
Max. instr. commited/clock	3
Instructions in reorder buffer	40

2 integer functional units (FU), 1 floating point FU, 1 branch FU, 1 Load FU, 1 Store FU

Pentium 4

° Also translates from 80x86 to micro-ops
  • But translates before the Icache

° P4 has better branch predictor, more FUs
  • Overclocked integer ALUs, so 4 integer ALU ops/cycle in the core

° Clock rates:
  • Pentium III 1 GHz v. Pentium IV 1.5 GHz
  • 10 stage pipeline vs. 20 stage pipeline

° Faster memory bus: 400 MHz v. 133 MHz

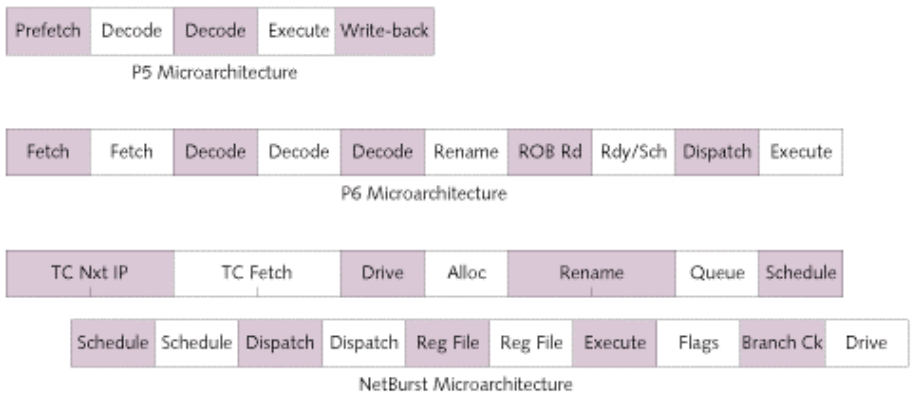
° Caches
  • Pentium III: L1I 16KB, L1D 16KB, L2 256 KB
  • Pentium 4: L1I 12 K μOps, L1D 8 KB, L2 256 KB
  • AMD Athlon: L1I 64KB, L1D 64KB, L2 256 KB
  • AMD Duron: L1I 64KB, L1D 64KB, L2 64 KB victim cache
  • Block size: PIII 32B v. P4 128B

Pentium 4 features

° Multimedia instructions 128 bits wide vs. 64 bits wide => 144 new instructions
  • When used by programs??
  • Still too short to be attractive compiler targets
    - Is a fair amount of compiler lore on compiling to vector architectures
  • Many of the new instructions are surprisingly slow

° Instruction Cache holds micro-operations vs. 80x86 instructions
  • no decode stages of 80x86 on cache hit
  • called "trace cache" (TC)

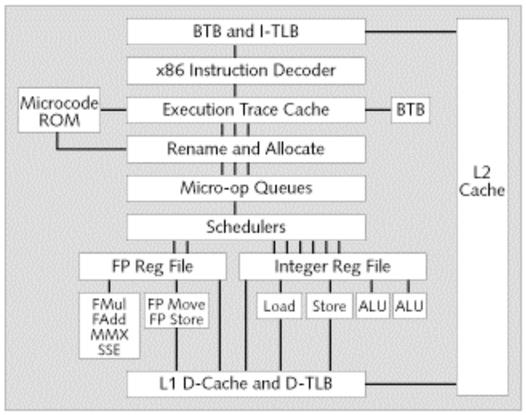
Pentium, Pentium Pro, Pentium 4 Pipeline



- Pentium (P5) = 5 stages
  Pentium Pro, II, III (P6) = 10 stages
  Pentium 4 (NetBurst) = 20 stages

"Pentium 4 (Partially) Previewed," Microprocessor Report, 8/28/00

Block Diagram of Pentium 4 Microarchitecture



- BTB = Branch Target Buffer (branch predictor)
- I-TLB = Instruction TLB, Trace Cache = Instruction cache
- RF = Register File; AGU = Address Generation Unit
- "Double pumped ALU" means ALU clock rate 2X => 2X ALU F.U.s

Pentium III v. Pentium 4 in benchmarks

- PC World magazine, Nov. 20, 2000
  - WorldBench 2000 benchmark (business)
  - P4 score @ 1.5 GHz: 164 (bigger is better)
  - PIII score @ 1.0 GHz: 167
  - AMD Althon @ 1.2 GHz: 180
  - (Media apps do better on P4 v. PIII)
- P4 has the marketing megahertz, but is actually down on performance

Why?

- Instruction count is the same for x86
- Clock rates: P4 > Althon > PIII
- How can P4 be slower?
- Time =
  Instruction count x CPI x 1/Clock rate
- Average Clocks Per Instruction (CPI) of P4 must be worse than Althon, PIII