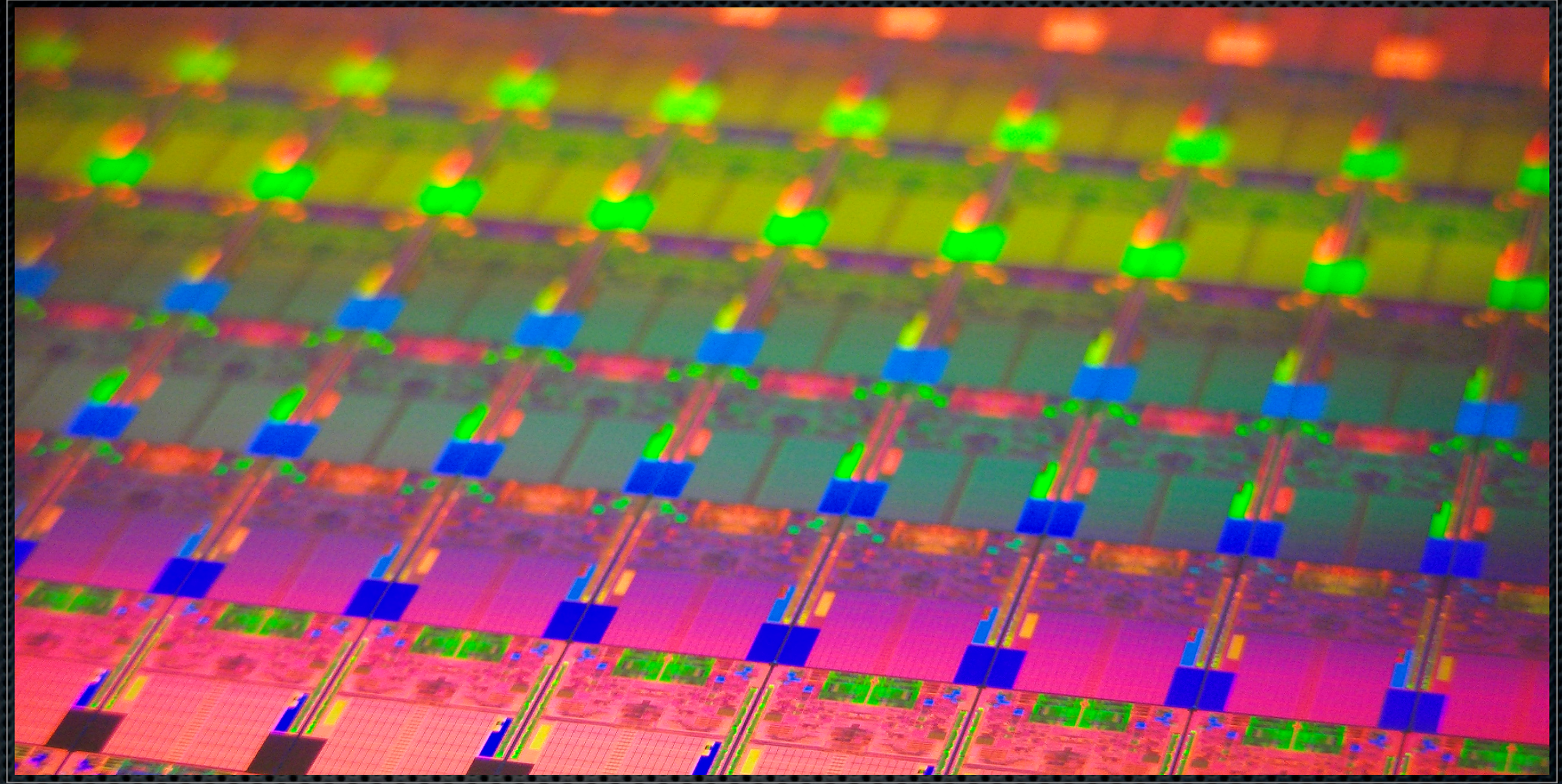

Introduction to Assembler

Andy Keep

Assembly Language

- What is assembly language?
  - Roughly one-to-one mapping to machine code
  - Provides full access to processor
- Why do we still need it?
  - Low-level operations, OS, and driver development
  - Performance sensitive tasks, numeric libraries
  - Generating from higher level languages, compilers

RISC vs. CISC

RISC vs. CISC

- RISC
  - Simple instructions
  - Load/Store
  - Register only operands
  - Fixed size instructions
- CISC
  - Complex instructions
  - Memory operands
  - Varying size instructions

RISC vs. CISC

Adding Two Integers From Memory

ARM7

```
mov  r1, #0
ldr r2, [r0,#8]
ldr r3, [r0,#12]
add r4,r2,r3
```

Intel x86

```
movl $0, %eax
addl 8(%ebp), %eax
addl 12(%ebp), %eax
```

RISC Instructions (ARM 7)

ADC	EOR	MSR	STR
ADD	LDC	MUL	SUB
AND	LDM	MVN	SWI
B	LDR	ORR	SWP
BIC	MCR	RSB	TEQ
BX	MLA	RSC	TST
CDP	MOV	SBC	
CMN	MRC	STC	
CMP	MRS	STM	

x86/x86_64 Instruction Set

AAA
AAD
AAM
AAS
ADC
ADD
ADDPD
ADDPS
ADDSD
ADDSS
ADDSUBPD
ADDSUBPS
AND
ANDPD
ANDPS
ANDNPD
ANDNPS
ARPL
BLENDPD
BLENDPS
BLENDVPD
BLENDVPS
BOUND
BSF
BSR
BSWAP
BT
BTC
BTR
BTS
CALL
CBW
CWDE
CDQE
CLC
CLD
CLFLUSH
CLI
CLTS
CMC
CMOVcc
CMP
CMPPD
CMPPS
CMPS
CMPSB
CMPSW
CMPSD
CMPSQ
CMPSD
CMPSS
CMPXCHG
CMPXCHG8B
CMPXCHG16B
COMISD
COMISS
CPUID
CRC32
CVTDQ2PD
CVTDQ2PS
CVTPD2DQ
CVTPD2DQ
CVPD2PI
CVTPD2PS
CVTPI2PD
CVTPI2PS
CVTPS2DQ
CVTPS2PD
CVTPS2PI
CVTSD2SI
CVTSD2SS
CVTSI2SD
CVTSI2SS
CVTSS2SD
CVTSS2SI
CVTTPD2DQ
CVTTPD2PI
CVTTPS2DQ
CVTTPS2PI
CVTTSD2SI
CVTTSS2SI
CWD
CDQ
CQO
DDA
DAS
DEC
DIV
DIVPD
DIVPS
DIVSD
DIVSS
DPPD
DPPS
EMMS
ENTER
EXTRACTPS
F2XM1
FABS
FADD
FADDP
FIADD
FBLD
FBSTP
FCHS
FCLEX
FNCLEX
FCMOVcc
FCOMI
FCOMIP
FUCOMI
FUCOMIP
FCOS
FDECSTP
FDECSTP
FDIV
FDIVP
FDIV
FDIVR
FDIVRP
FIDIVR
FFREE
FICOM
FICOMP
FILD
FINCSTP
FINIT
FNINIT
FIST
FISTP
FISTTP
FLD
FLD1
FLDL2T
FLDL2E
FLDPI
FLDLG2
FLDLN2
FLDZ
FLDCW
FLDENV
FMUL
FMULP
FIMUL
FNOP
FPATAN
FPREM
FPREM1
FPTAN
FRNDINT
FRSTOR
FSAVE
FNSAVE
FSCALE
FSIN
FSINCOS
FSQRT
FST
FSTP
FSTCW
FNSTCW
FSTENV
FNSTENV
FSTSW
FNSTSW
FSUB
FSUBP
FISUB
FSUBR
FSUBRP
FISUBR
FTST
FUCOM
FUCOMP
FUCOMPP
FXAM
FXCH
FXRSTOR
FXSAVE
FXTRACT
FYL2X
FYL2XP1
HADDPD
HADDPS
HLT
HSUBPD
HSUBPS
IDIV
IMUL
IN
INC
INS
INSB
INSW
INSD
INSERTPS
INT
INVD
INVLPG
IRET
IRETD
Jcc
JMP
LAHF
LAR
LDDQU
LDMXCSR
LDS
LES
LFS
LGS
LSS
LEA
LEAVE
LFENCE
LGDT
LIDT
LLDT
LMSW
LOCK
LODS
LODSB
LODSW
LODSD
LODSQ
LOOP
LOOPcc
LSL
LTR
MASKMOVDQU
MASKMOVQ
MAXPD
MAXPS
MAXSD
MAXSS
MFENCE
MINPD
MINPS
MINSD
MINSS
MONITOR
MOV
MOVAPD
MOVAPS
MOVBE
MOVD
MOVQ
MOVDDUP
MOVDQA
MOVDQU
MOVDQ2Q
MOVHLPS
MOVHPD
MOVHPS
MOVLHPS
MOVLPD
MOVLPS
MOVMSKPD
MOVMSKPS
MOVNTDQA
MOVNTDQ
MOVNTI
MOVNTPD
MOVNTPS
MOVNTQ
MOVQ
MOVQ2DQ
MOVS
MOVSB
MOVSW
MOVSD
MOVSQ
MOVSHDUP
MOVSLDUP
MOVSS
MOVSX
MOVSXD
MOVUPD
MOVUPS
MOVZX
MPSADBW
MUL
MULPD
MULPS
MULSD
MULSS
MWAIT
NEG
NOP
NOT
OR
ORPD
ORPS
OUT
OUTS
OUTSB
OUTSW
OUTSD
PABSB
PABSW
PABSD
PACKSSWB
PACKSSDW
PACKUSDW
PACKUSWB
PADDB
PADDW
PADDD
PADDQ
PADDSB
PADDSW
PADDUSB
PADDUSW
PALIGNR
PAND
PANDN
PAUSE
PAVGB
PAVGW
PBLENDVB
PBLENDW
PCMPEQB
PCMPEQW
PCMPEQD
PCMPEQQ
PCMPESTRI
PCMPESTRM
PCMPISTRI
PCMPISTRM
CPMGTB
PCMPGTW
PCMPGTD
PCMPGTQ
PEXTRB
PEXTRD
PEXTRQ
PEXTRW
PHADDW
PHADDD
PHADDSW
PHMINPOSUW
PHSUBW
PHSUBD
PHSUBSW
PINSRB
PINSRD
PINSRQ
PINSRW
PMADDUBSW
PMADDWD
PMAXSB
PMAXSD
PMAXSW
PMAXUB
PMAXUD
PMAXUW
PMINSB
PMINSD
PMINSW
PMINUB
PMINUD
PMINUW
PMOVMSKB
PMOVSX
PMOVZX
PMULDQ
PMULHRSW
PMULHUW
PMULHW
PMULLD
PMULLW
PMULUDQ
POP
POPA
POPAD
POPCNT
POPF
POPFD
POPFQ
POR
PREFETCHh
PSADBW
PSHUFB
PSHUFD
PSHUFHW
PSHUFLW
PSHUFW
PSIGNB
PSIGNW
PSIGND
PSLLDQ
PSLLW
PSLLD
PSLLQ
PSRAW
PSRAD
PSRLDQ
PSRLW
PSRLD
PSRLQ
PSUBB
PSUBW
PSUBD
PSUBQ
PSUBSB
PSUBSW
PSUBUSB
PSUBUSW
PTEST
PUNPCKHWB
PUNPCKHWD
PUNPCKHDQ
PUNPCKHQDQ
PUNPCKLBW
PUNPCKLWD
PUNPCKLDQ
PUNPCKLQDQ
PUSH
PUSHA
PUSHAD
PUSHF
PUSHFD
PXOR
RCL
RCR
ROL
ROR
RCPPS
RCPSS
RDMSR
RDPMC
RDTSC
RDTSCP
REP
REPE
REPZ
REPNE
REPNZ
RET
ROUNDPD
ROUNDPS
ROUNDSD
ROUNDSS
RSM
RSQRTPS
RSQRTSS
SAHF
SAL
SAR
SHL
SHR
SBB
SCAS
SCASB
SCASW
SCASD
SETcc
SFENCE
SGDT
SHLD
SHRD
SHUFPD
SHUFPS
SIDT
SLDT
SMSW
SQRTPS
SQRTSD
SQRTSS
STC
STD
STI
STMXCSR
STOS
STOSB
STOSW
STOSD
STOSQ
STR
SUB
SUBPD
SUBPS
SUBSD
SUBSSSWAPGS
SYSCALL
SYSENTER
SYSEXIT
SYSRET
TEST
UCOMISD
UCOMISS
UD2
UNPCKHPD
UNPCKHPS
UNPCKLPD
UNPCKLPS
VERR
VERW
WAIT
FWAIT
WBINVD
WRMSR
XADD
XCHG
XGETBV
XLAT
XLATB
XOR
XORPD
XORPS
XRSTOR
XSAVE
XSETBV

Intel x86 and x86_64 (AMD64)

A Long History

- 8086/8088 - 16-bit processors (1978)
- 80286 - Added protected mode (1982)
- 80386 - 32-bit processor (1985)
- 80486 - Integrated FPU (1989)
- Pentium - Superscalar, MMX (1993)
- P6 Family - Microarchitecture, SSE (1995-1999)
- Pentium 4 - Hyperthreading, SSE2, SSE3 (2000 -2006)
- Many more in modern product line, still compatible!

32-bit Mode Registers

- 32-bit: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
- 16-bit: AX, BX, CX, DX, SI, DI, SP, BP
- 8-bit: AH, BH, CH, DH, AL, BL, CL, DL
- Segment: CS, DS, SS, ES, FS, GS
- EFLAGS
- x87 FPU: ST0 - ST7, status, control, tag words, data op pointer, ip
- MMX: MM0 - MM7
- XMM: XMM0 - XMM7, MXCSR
- Control (CR0, CR2, CR3, CR4) system table (GDTR, LDTR, IDTR, task register)
- Debug: DR0, DR1, DR2, DR3, DR6 and DR7
- MSR
- Instruction Pointer: EIP

64-bit Mode Registers

- 64-bit: RAX, RBX, RCX, RDX, RSI, RDI, RSP, RBP, R8 - R15
- 32-bit: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, R8D - R15D
- 16-bit: AX, BX, CX, DX, SI, DI, SP, BP, R8W - R15W
- 8-bit: AL, BL, CL, DL, SIL, DIL, SPL, BPL, R8L - R15L (with REX prefix) AL, BL, CL, DL, AH, BH, CH, DH
- Segment: CS, DS, SS, ES, FS, GS
- RFLAGS
- x87 FPU: ST0 - ST7, status, control, tag words, data op pointer, ip
- MMX: MM0 - MM7
- XMM: XMM0 - XMM15, MXCSR
- Control (CR0, CR2, CR3, CR4, CR8) system table (GDTR, LDTR, IDTR, task register)
- Debug: DR0, DR1, DR2, DR3, DR6 and DR7
- MSR
- Instruction Pointer: RIP
- RDX:RAX register pair for 128-bit operand

Instruction Operands

- Register Operands
  - Register from the set available for a given subset of instructions
- Memory Operands
  - Include: displacement, base, index, and scale
  - Formatted: disp(base,index,scale)
- Generally, at most, one memory operand may be used

Application Binary Interface

- Dependent on language, operating system, mode
- C-Style, unix, 32-bit mode (cdecl)
  - Arguments push on stack left to right
  - Return value in EAX (or ST0 for floating point)
  - EAX, ECX, EDX available for function
- Different standards for windows or for C++

Application Binary Interface

- x86-64 System V (used on most unixes)
  - Integer Arguments: RDI, RSI, RDX, RCX, R8, R9
  - Floating Point Arguments: XMM0 - XMM7
  - All other arguments on stack
  - Return: RAX (Integer) XMM0 - XMM1 (Floating Point) ST0, ST1 (Long Double)
  - Callee-saved: RBX, RBP, R12 - R15
  - Many other details, enough to fill a book!

Tools of the trade

Tools of the Trade

- Assembler and Linker (GNU `as` and `ld`)
  - Stand alone development
  - Library Development
- Compiler (GCC)
  - Inline Assembler Use
  - Access C-library from Assembler
  - Educational Tool

Inline Assembler

- Assembly one-liners
  - `__asm__("movl %%eax, %%ecx")`
- More complex inline assembler
  - `__asm__(`*code lines*
    `:` *output operands*
    `:` *input operands*
    `:` *clobber list*`)`
- Double % used to differentiate registers from C arguments
- Clobber list tells C register allocator which registers are used

Inline Asm Operand Options

r	register
a	RAX, EAX, AX, AL
b	RBX, EBX, BX, BL
c	RCX, ECX, CX, CL
d	RDX, EDX, DX, DL
S	RSI, ESI, SI
D	RDI, EDI, DI
m	memory location
g	any location

Inline Assembler Example

```
int x = 7, y;
__asm__(“movl %0, %%eax;
        imull $9, %%eax;
        movl %%eax, %1;”
      : “=r” (y)
      : “r” (x)
      : “%eax”);
```

Inline Assembler Example (Optimized)

```
int x = 7, y;
__asm__(“imull $9, %%eax”
        : “=a” (y)
        : “a” (x));
```

Assembler with C Driver

- Program is written in C, declaring the function(s) to be called as an external function
- Assembler file is written exporting function as global
- Assembler function must obey C ABI
- C compiler is used to compile and link program

Assembler Language

- Sections: text, named, data, bss, absolute, undefined
- Directives: start with dot (.), indicate subsections, linker directives, assembler directives, macro definitions
- Functions exported with .global, must follow ABI
- GNU as uses AT&T style, with reversed operand syntax
- BSD unixes (including Mac OS X) prefix function labels with underscore (_)
- File extension is “s” (i.e. my_asm_lib.s)

C Driver, Compilation

```
#include <stdio.h>

long sum(long, long*);

int main(int argc, int * argv[]) {
  long v[] = {8, 13, 15, 4, 32};
  printf("sum is: %ld\n", sum(5,v));
  return 0;
}

gcc -m64 -o sum sum.s sum.c
```

Assembler Example

```
        .text
        .globl _sum
_sum:   movq $0, %rcx
        movq $0, %rax
L0:     cmpq %rdi, %rcx
        jge L1
        addq (%rsi,%rcx,8), %rax
        addq $1, %rcx
        jmp L0
L1:     ret
```

Learning from the Compiler

- The C Compiler can be instructed to stop compilation at the assembler stage by using `-S` argument
- Optimized code can be difficult to untangle
- Turn off the optimizer with `-O0`
- C Compiler uses frame pointer and stack pointer

A Simple Example

```
#include <stdio.h>

int main (int argc, char * argv[]) {
   printf(“Hello, world!\n”);
   return 0;
}

gcc -m64 -S hello.c
```

A Simple Example

```
    .cstring
LC0:
    .ascii "Hello, world!\0"
    .text
.globl _main
_main:
LFB3:
    pushq %rbp
LCFI0:
    movq  %rsp, %rbp
LCFI1:
    subq  $16, %rsp
LCFI2:
    movl  %edi, -4(%rbp)
    movq  %rsi, -16(%rbp)
    leaq  LC0(%rip), %rdi
    call  _puts
    movl  $0, %eax
    leave
    ret

/* -- Truncated here! -- */
```

The Floating Point Unit

- Originally separate chip, FPU
- Has separate instructions, registers, and IP
- SSE instructions may be used in place of FPU on modern processors, especially in 64-bit mode
- 64-bit mode passes most FPU arguments to function calls through XMM0-XMM7 (SSE) registers

SIMD x86 Extensions

- MMX - Added 64-bit registers with packed integer data types
- SSE - Added 128-bit registers with packed integer data types
- SSE2 - Extended SSE with packed floating point types
- SSE3, SSSE3, SSE4 - Added 8 registers for operations on 64-bit processors and new instructions in each revision
  - Newest standards are only partially supported by AMD

SSE Example

```
        .text
        .globl _fsum
_fsum:  movq $0, %rcx
        subsd %xmm0, %xmm0
L0:     cmpq %rdi, %rcx
        jge L1
        movsd (%rsi, %rcx, 8), %xmm1
        addsd %xmm1, %xmm0
        addq $1, %rcx
        jmp L0
L1:     ret
```

Intel Architecture

Complex Architecture

- Assembler instructions converted into µops
- Out of order execution, internal scheduler
- Hyper-threading
- Multiple ALUs
- Separate Load and Store units, with simultaneous load and store to different memory locations
- Branch prediction unit

Thanks

Questions?

References

- Intel References
  - http://www.intel.com/products/processor/manuals/
- AMD References
  - http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_739_7044,00.html
- GNU GCC/GAS
  - http://gcc.gnu.org/
  - http://www.gnu.org/software/binutils/
  - http://sourceware.org/binutils/docs/as/index.html