Compilers and Libraries

Staffan Tjernström

Lead Software Engineer

Eagle Seven Technologies, LLC

stjernstrom@eagleseven.com

https://www.linkedin.com/pub/staffan-tjernstr%C3%B6m/6/595/137

Why we care

with apologies to Sir Terrance Pratchett, OBE

Knowledge

Why we care

with apologies to Sir Terrance Pratchett, OBE

Knowledge

= Power

= Energy

= Mass

A computer is just a genteel black hole that knows how to count

Libraries

- Doesn't know how it will be used
- Has to perform as well as possible in all circumstances
- Only allowed very tightly defined side-effects if any
- Very tightly controlled interfaces

These are somewhat contradictory!

The classic example

Thread 1

```
while( running )
{
    int item_count(0);
    for( auto i : *p_map )
    {
        item_count++;
    }
    p_map->erase( item_count - 1 );
}
```

Thread 2

```
for( int i = 0; i < 1024 * 1024 ; i++ )
{
    my_map[ i ] = i * 16;
}
running = false;
```

Occasional SegVio!

Legalese

Example Code

```
#include <stdio.h>
int main( )
{
    char const text[] = “Hello Universe”;
    FILE *fp = fopen(“invalid/path/to/file”, “w+”);
    fwrite( text, sizeof( char ), sizeof( text ),  fp );
    fclose( fp );
}
```

Segmentation Fault!

fwrite

```
_IO_size_t
_IO_fwrite (buf, size, count, fp)
     const void *buf;
     _IO_size_t size;
     _IO_size_t count;
     _IO_FILE *fp;
{
  _IO_size_t request = size * count;
  _IO_size_t written = 0;
  CHECK_FILE (fp, 0);
  if (request == 0)
    return 0;
  _IO_acquire_lock (fp);
  if (_IO_vtable_offset (fp) != 0 || _IO_fwide (fp, -1) == -1)
    written = _IO_sputn (fp, (const char *) buf, request);
  _IO_release_lock (fp);
  if (written == request || written == EOF)
    return count;
  else
    return written / size;
}
```

```
static inline void
__attribute__ ((__always_inline__))
_IO_acquire_lock_fct (_IO_FILE **p)
{
  _IO_FILE *fp = *p;
  if ((fp->_flags & _IO_USER_LOCK) == 0)
    _IO_funlockfile (fp);
}
```

Multiplatform Considerations

- Different type for same argument
  - int vs size_t
  - 32 vs 64-bit
- Different fundamental behaviour
  - ASIO vs Overlapped
  - WaitForMultipleObjects vs parameterized callbacks

Library Contracts

- Users need to be aware of the (lack of) guarantees libraries offer
- Some guarantees have distinct performance effects
- Storage limitations

Successful implementations

- CP/M 3 BIOS jump vectors
- glibc
- Boost
- C++ standard library
- Linux syscall()

- Many more

Compilers

- Processors have complex behaviours
- System components no longer equal in latency
- Target systems not the same as build systems
- Virtual Targets

Loopy code

```
extern int elem[64];

int i, sum;
for( i = 63; i; i-- )
{
    sum = sum + elem[ i ];
}
```

1979 version

```
LDA $elem
STA #00
LDA $0000
LDY $003F
LOOP ADC (#00), Y
DEY
BEQ END
JMP LOOP
END
```

1979 Behaviour

- Linear complexity up to 255 bytes
- Easy to deduce C from assembler

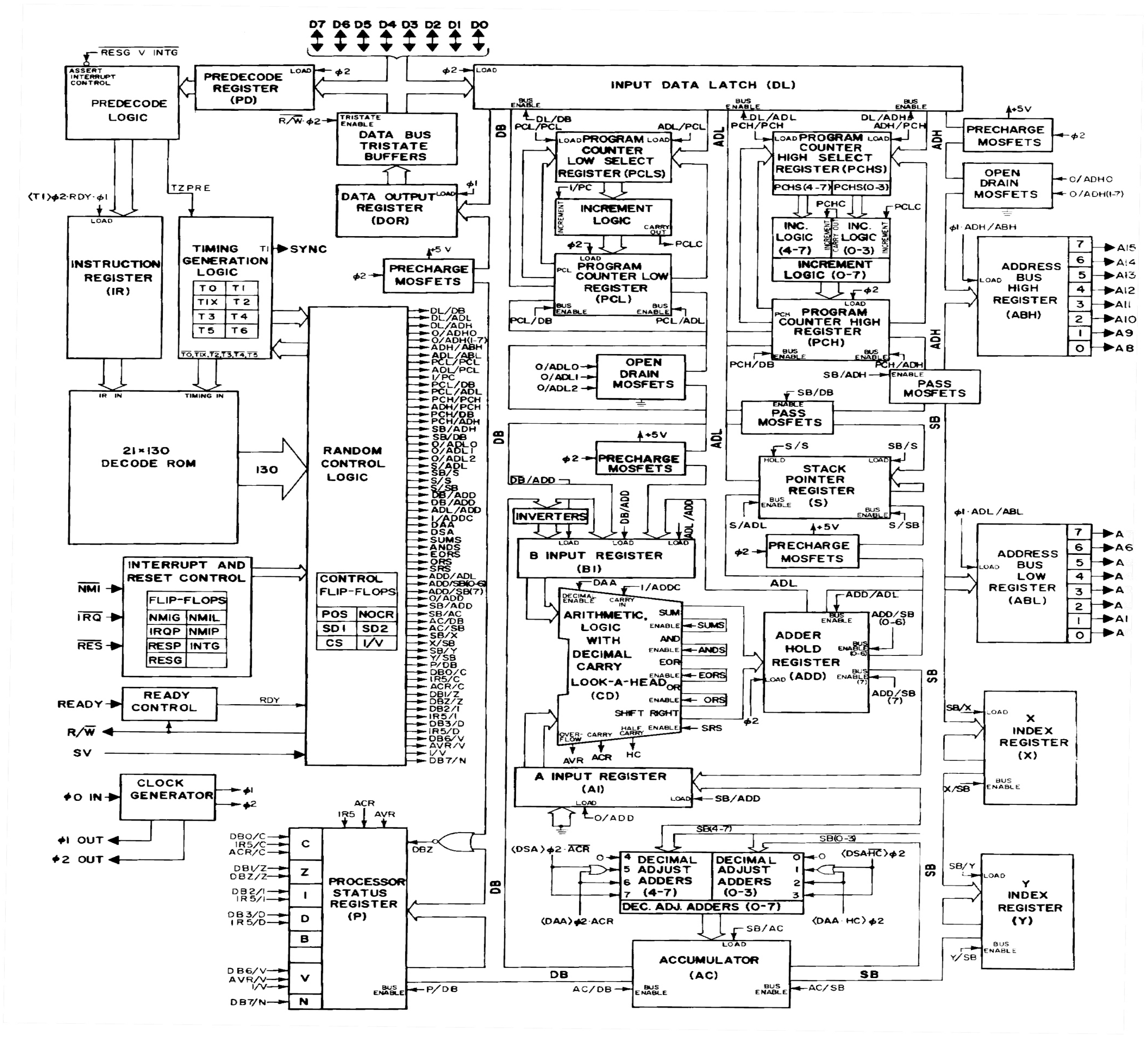
2014 version

```
40065e:       movl   $0x0,-0x8(%rbp)
400665:       movl   $0x3f,-0x4(%rbp)
40066c:       jmp    400681 <main+0x2b>
40066e:       mov    -0x4(%rbp),%eax
400671:       cltq
400673:       mov    elem(,%rax,4),%eax
40067a:       add    %eax,-0x8(%rbp)
40067d:       subl   $0x1,-0x4(%rbp)
400681:       cmpl   $0x0,-0x4(%rbp)
400685:       jne    40066e <main+0x18>
```

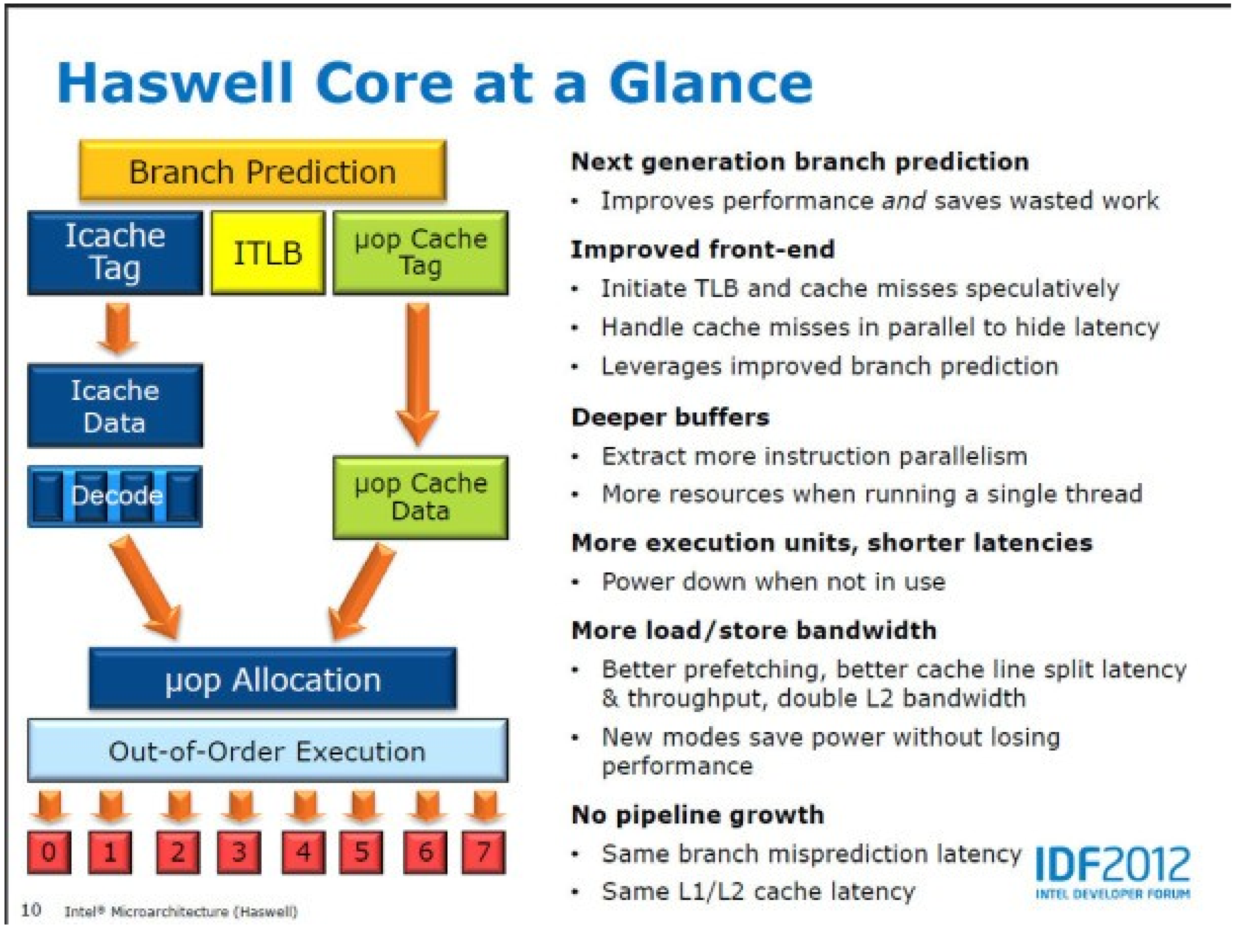
2014 behaviour

- Still possible to deduce C from code
- Not linear behaviour
  - Each run takes different amount of time !

Typical 1979 processor


D7 D6 D5 D4 D3 D2 D1 D0
INPUT DATA LATCH (DL)
PREDECODE REGISTER (PD)
PREDECODE LOGIC
DATA BUS TRISTATE BUFFERS
DATA OUTPUT REGISTER (DOR)
INSTRUCTION REGISTER (IR)
TIMING GENERATION LOGIC
SYNC
21×130 DECODE ROM
130
RANDOM CONTROL LOGIC
CONTROL FLIP-FLOPS
INTERRUPT AND RESET CONTROL
NMI
IRQ
RES
READY
READY CONTROL
R/W
SV
CLOCK GENERATOR
φ0 IN
φ1 OUT
φ2 OUT
PROCESSOR STATUS REGISTER (P)
PROGRAM COUNTER LOW SELECT REGISTER (PCLS)
INCREMENT LOGIC
PROGRAM COUNTER LOW REGISTER (PCL)
PROGRAM COUNTER HIGH SELECT REGISTER (PCHS)
PROGRAM COUNTER HIGH REGISTER (PCH)
PRECHARGE MOSFETS
OPEN DRAIN MOSFETS
PASS MOSFETS
STACK POINTER REGISTER (S)
ADDRESS BUS HIGH REGISTER (ABH)
ADDRESS BUS LOW REGISTER (ABL)
A15 A14 A13 A12 A11 A10 A9 A8
INVERTERS
B INPUT REGISTER (BI)
ARITHMETIC LOGIC WITH DECIMAL CARRY LOOK-A-HEAD (CD)
A INPUT REGISTER (AI)
ADDER HOLD REGISTER (ADD)
X INDEX REGISTER (X)
Y INDEX REGISTER (Y)
DECIMAL ADJUST ADDERS (4-7)
DECIMAL ADJUST ADDERS (0-3)
DEC. ADJ. ADDERS (0-7)
ACCUMULATOR (AC)
DB
ADL
ADH
SB

A single Haswell core



Haswell Core at a Glance

Next generation branch prediction

- Improves performance *and* saves wasted work

Improved front-end

- Initiate TLB and cache misses speculatively
- Handle cache misses in parallel to hide latency
- Leverages improved branch prediction

Deeper buffers

- Extract more instruction parallelism
- More resources when running a single thread

More execution units, shorter latencies

- Power down when not in use

More load/store bandwidth

- Better prefetching, better cache line split latency & throughput, double L2 bandwidth
- New modes save power without losing performance

No pipeline growth

- Same branch misprediction latency
- Same L1/L2 cache latency

IDF2012 INTEL DEVELOPER FORUM

2014 version

```
40065e:    movl   $0x0,-0x8(%rbp)
400665:    movl   $0x3f,-0x4(%rbp)
40066c:    jmp    400681 <main+0x2b>
40066e:    mov    -0x4(%rbp),%eax
400671:    cltq
400673:    mov    elem(,%rax,4),%eax
40067a:    add    %eax,-0x8(%rbp)
40067d:    subl   $0x1,-0x4(%rbp)
400681:    cmpl   $0x0,-0x4(%rbp)
400685:    jne    40066e <main+0x18>
```

Loop Carry Dependency

- Address update is done using the result of a previous loop iteration
- Deprives the out-of-order engine of the capability to have multiple iterations in-flight
- Memory loads, instruction latency, bypass delays cannot be amortized efficiently.

Everyone is writing parallel programs

Optimzed Straight Line loopiness

```
400510:    pshufd $0x1b,0x200c47(%rip),%xmm1      # 601160 <elem+0xe0>
400519:    xor    %eax,%eax
40051b:    pshufd $0x1b,0x200c4c(%rip),%xmm0      # 601170 <elem+0xf0>
400524:    paddd  %xmm1,%xmm0
400528:    pshufd $0x1b,0x200c1f(%rip),%xmm1      # 601150 <elem+0xd0>
400531:    paddd  %xmm1,%xmm0
#
# pshufd / paddd repeated through elem+0x00
#
4005d1:    movdqa %xmm1,%xmm2
4005d5:    psrldq $0x8,%xmm2
4005da:    paddd  %xmm2,%xmm1
4005de:    movdqa %xmm1,%xmm3
4005e2:    psrldq $0x4,%xmm3
4005e7:    paddd  %xmm3,%xmm1
4005eb:    movdqa %xmm1,%xmm4
4005ef:    movd   %xmm4,0xc(%rsp)
4005f5:    mov    0xc(%rsp),%edx
4005f9:    add    0x200a8d(%rip),%edx      # 60108c <elem+0xc>
4005ff:    add    0x200a83(%rip),%edx      # 601088 <elem+0x8>
400605:    add    0x200a79(%rip),%edx      # 601084 <elem+0x4>
```

Store Forwarding

- Use data written earlier without going through memory or cache
- Data size and alignment in the store and load must follow restrictive rules
  - A store forwarded load must have the same start point as the forwarded store.
  - The length of the load must be less than or equal to the length of the store.
- Data from the store must be available by the time the load starts

Store Forward example

```
mov [EBP],'helloworld'
mov AL, [EBP]          ;not blocked
mov BL, [EBP+1]        ;blocked
mov CL, [EBP+2]        ;blocked
mov DL, [EBP+3]        ;blocked
mov AL, [EBP]          ;not blocked, issued
                       before blocked loads
```

Suggested Reading

- Intel® 64 and IA-32 Architectures Optimization Reference Manual
- The Software Optimization Cookbook

We're Hiring!

http://www.eagleseven.com/candidates.php