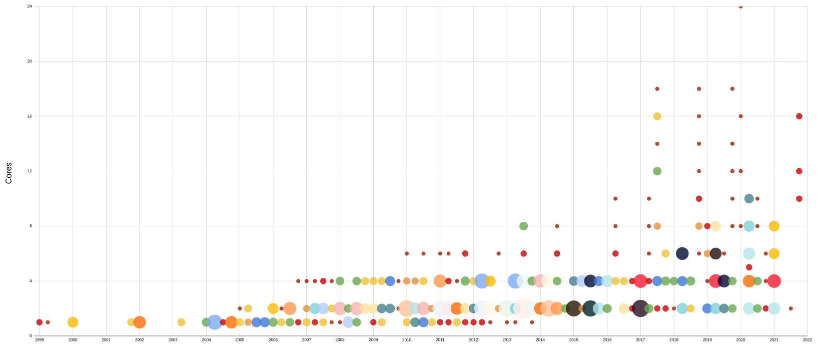
Program Optimization – modern CPUs

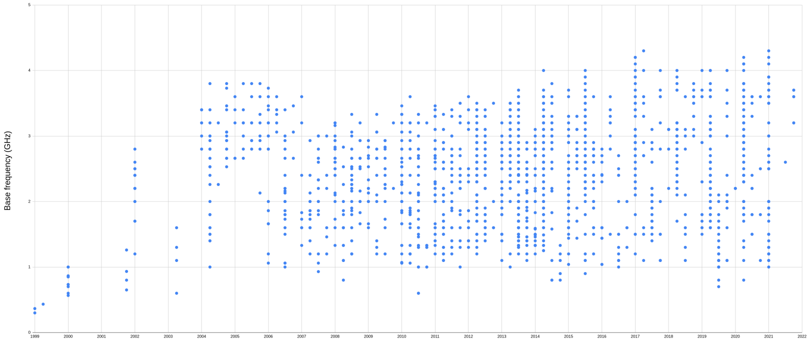
simon.marechal@synacktiv.com

2023/2024

Amount of cores of Intel consumer CPUs



Base frequency of Intel consumer CPUs



Programming the modern PDP-11

- number of transistors increase exponentially, but processor frequency does not;
- how to increase performance without changing the computing paradigm?
- CPU builders cheat!
  - multiple parallel execution units ;
  - deep pipelining ;
  - and ... eager execution
    - many exploitable bugs!

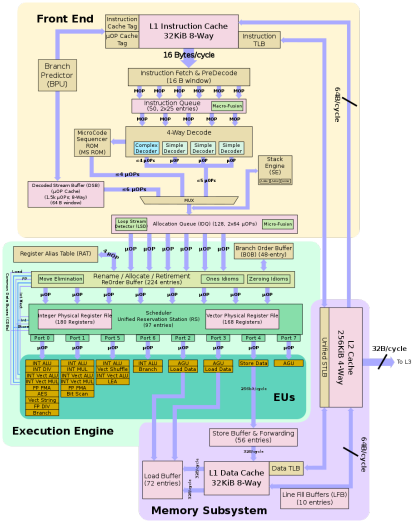

Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50, 2x25 entries)
Macro-Fusion
MicroCode Sequencer ROM (MS ROM)
4-Way Decode
Complex Decoder
Simple Decoder
≤4 µOPs
µOP
Stack Engine (SE)
≤5 µOPs
Decoded Stream Buffer (DSB) (µOP Cache) (1.5k µOPs; 8-Way) (64 B window)
≤6 µOPs
MUX
Loop Stream Detector (LSD)
Allocation Queue (IDQ) (128, 2x64 µOPs)
Micro-Fusion
64B/cycle
Register Alias Table (RAT)
4 µOP
Branch Order Buffer (BOB) (48-entry)
Move Elimination
Rename / Allocate / Retirement ReOrder Buffer (224 entries)
Ones Idioms
Zeroing Idioms
Common Data Buses (CDBs)
Load
FP
Int Vect
Int
Store
Integer Physical Register File (180 Registers)
Scheduler Unified Reservation Station (RS) (97 entries)
Vector Physical Register File (168 Registers)
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
INT ALU
INT DIV
INT Vect ALU
INT Vect MUL
FP FMA
AES
Vect String
FP DIV
Branch
INT MUL
Bit Scan
Vect Shuffle
LEA
AGU
Load Data
Store Data
256bit/cycle
EUs
Execution Engine
Unified STLB
L2 Cache 256KiB 4-Way
32B/cycle
To L3
Store Buffer & Forwarding (56 entries)
32B/cycle
Load Buffer (72 entries)
L1 Data Cache 32KiB 8-Way
Data TLB
Line Fill Buffers (LFB) (10 entries)
Memory Subsystem

Figure 1: :'(

CPU architecture

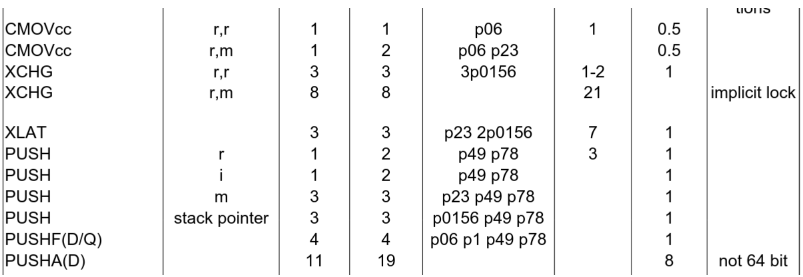
Micro ops

The lowest level programming language available to the programmer is assembly. However, this is a lie!

- assembly instructions are converted to a lower level, undocumented, language: micro ops
- CPUs have several parallel execution ports, virtual registers, etc.
- micro ops can in turn be optimized

(Such knowledge is important for low level optimization, but can be ignored by normal humans)

Execution ports



							tions
CMOVcc	r,r	1	1	p06	1	0.5	
CMOVcc	r,m	1	2	p06 p23		0.5	
XCHG	r,r	3	3	3p0156	1-2	1	
XCHG	r,m	8	8		21		implicit lock
XLAT		3	3	p23 2p0156	7	1	
PUSH	r	1	2	p49 p78	3	1	
PUSH	i	1	2	p49 p78		1	
PUSH	m	3	3	p23 p49 p78		1	
PUSH	stack pointer	3	3	p0156 p49 p78		1	
PUSHF(D/Q)		4	4	p06 p1 p49 p78		1	
PUSHA(D)		11	19			8	not 64 bit

Figure 2: Excerpt from
https://www.agner.org/optimize/instruction_tables.pdf

Pipelines

In order to increase throughput, instructions are fed to a pipeline where operations happen in turn:

- ancient, classical, pipeline: instruction fetch, instruction decode, execute, memory access, write back
- Pentium 4 CPUs had a 31-stage pipeline, at the same time AMD64 CPUs had a 12-stage pipeline
- must be kept as full as possible to maintain performance

Out of order execution, register rewriting

Modern CPUs can turn this ...

```
mov eax, [mem1]
imul eax, 6
mov [mem2], eax
mov eax, [mem3]
add eax, 2
mov [mem4], eax
```

... into this!

```
mov eax, [mem1]
mov ebx, [mem3]
add ebx, 2
imul eax, 6
mov [mem2], eax
mov [mem4], ebx
```

Data dependency

```c
double list[100];
double sum = 0.;
for (int i=0; i<100; i++)
  sum += list[i];
```

```c
double list[100];
double sumA = 0., sumB = 0.;
double sumC = 0., sumD = 0.;
for (int i=0; i<100; i+=4) {
  sumA += list[i+0];
  sumB += list[i+1];
  sumC += list[i+2];
  sumD += list[i+3];
}
double sum = sumA + sumB +
             sumC + sumD;
```

Branch predictors and eager evaluation

- to maximize performance, pipelines must be kept full
- what happens on a conditional branch?
- CPUs *guess*, and start on a branch
- rollback if necessary
  - can be observable!

Intel guidelines

Static prediction:

- predict unconditional branches to be taken
- predict indirect branches to be NOT taken
- predict backward conditional branches to be taken; rule is suitable for loops
- predict forward conditional branches to be NOT taken

Dynamic prediction:

- undocumented, you are on your own :)

Memory hierarchy

Levels of memory

- registers
- caches
- RAM
- storage
- network?

There is a **huge** performance gap between each category!

Most important consideration!

- a jump is not as bad as an uncached memory read;
- maximizing throughput means avoiding latency: cache aware algorithms, pre-fetching, etc.
- knowledge of micro-ops is pretty important.

Note on *C is a portable assembler*

It is an assembler for a **very** abstract PDP-11.

CPU concepts

- registers
- virtual memory
- execution ports
- cache
- hyper-threading
- SIMD
- ...

C concepts

- variable types
- pointers
- undefined behavior
- sequence points
- aliasing
- `struct` and `union` types
- ...

Task

Optimize `simple-c/main.c`

- remove the function calls
- remove useless operations

QUESTIONS?

?

Thank you for your attention

SYNACKTIV DIGITAL SECURITY