Review Paper on AMD’s Ryzen Zen architecture based Processor

ECS409 Project submitted to Dr Aditya Sankar Medury

K Rohit Taeja, Dhawal Patil, H Dinesh Adhithya, Mehul Paithane, Kiran D

Indian Institute of Science Education and Research, Bhopal

Table of contents

1. Introduction, History and Evolution

2. Processor Architecture, Structure and Classification

3. Instruction Set Architecture

4. Applications

Introduction, History and Evolution

AMD and Ryzen

- 1999 - AMD64 was introduced as a competition to Intel’s IA-64.
- 2016 - Zen-based preview system was demonstrated near Intel Developer Forum 2016.
- 2017 - Commercial launch of Ryzen series processors.
- Ryzen chips are cooler, faster and consume less power than FX chips.

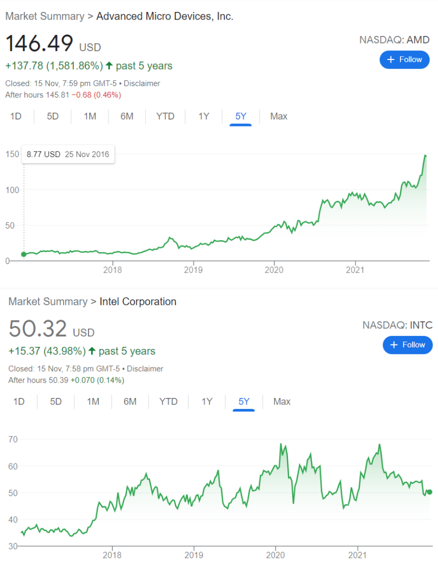
Ryzen’s Competition

- IPC uplift was 52% higher than Excavator but lesser than Intel’s Kaby Lake.
- Ryzen compensated by offering more cores.
- Power consumption and heat emission were competitive with Intel.
- Multi-threaded performance at relatively same price point of Intel’s Core processors.

The Rise and Rise of Ryzen

- AMD’s CPU market share has increased.
- Intel’s CPU market share has dropped drastically.
- Ryzen was developed under AMD’s new CEO Lisa Su.
- AMD share price has rocketed after Ryzen.
- Intel share price has been stagnant.

AMD vs Intel Share Price



Market Summary > Advanced Micro Devices, Inc.

146.49 USD

NASDAQ: AMD

+137.78 (1,581.86%) ↑ past 5 years

Closed: 15 Nov, 7:59 pm GMT-5 • Disclaimer

After hours 145.81 −0.68 (0.46%)

8.77 USD 25 Nov 2016

Market Summary > Intel Corporation

50.32 USD

NASDAQ: INTC

+15.37 (43.98%) ↑ past 5 years

Closed: 15 Nov, 7:58 pm GMT-5 • Disclaimer

After hours 50.39 +0.070 (0.14%)

Figure 1: AMD vs Intel Share Price

AMD vs Intel CPU Market Share

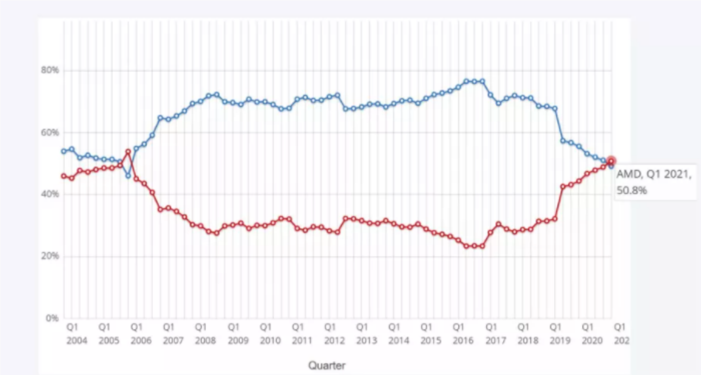
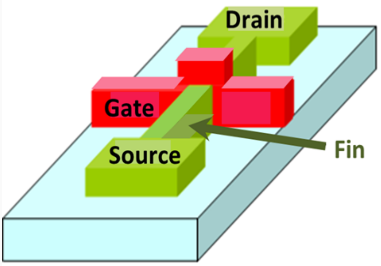


Figure 2: AMD vs Intel Market Share

Processor Architecture, Structure and Classification

Fabrication Process

FinFET Transistors



- Power efficient and sensitive
- Faster switching time
- 14nm technology , replaced 28 nm used in excavator.

Memory Hierarchy

The Zen based processors support DDR4 memory up to 8 channels and Error Correction Code(ECC). Cache Organization

- L0μOP cache: 2,048μOPs, 8-way set associative (larger than excavator)
- L1I Cache: 64 KiB 4-way set associative
- L1D Cache: 32 KiB 8-way set associative
- L2 Cache: 512 KiB 8-way set associative
- L3 Cache: Victim Cache
- L1 Cache follows write back policy. (excavator had write through).
- L1 and L2 2x faster
- L3 5X faster
- L1 and L2 dedicated to each core , L3 shared across cores.

Core Engine

- Each core decode four instructions per clock cycle and has an micro-op (simple internal representation of an architectural instruction) cache that feeds into two schedulers one for floating point and other for integer segments.
- Two threads per core (Simultaneous Multithreading)
- Improved branch prediction
- Large Op cache
- Larger Instruction Schedulers

Pipeline

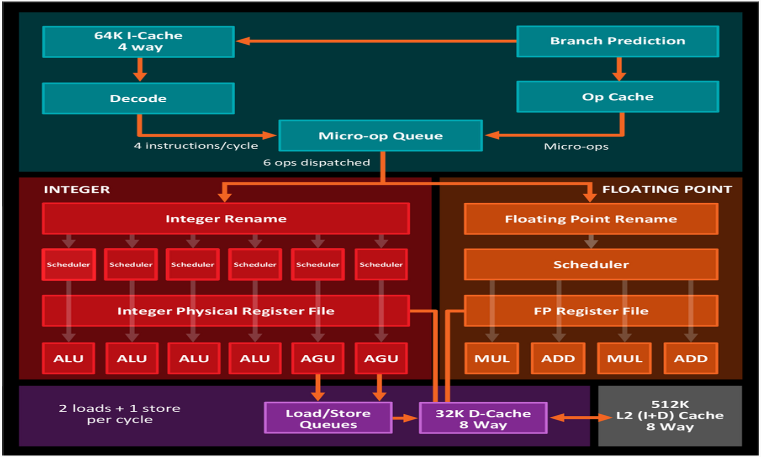
Each Ryzen core has 1 Floating-point unit and 1 integer-unit.

- Each Integer unit has 6 pipes, 4 ALUs (Arithmetic Logic Unit) and 2 AGUs (Address Generation Unit).
- These AGUs can perform two 16-byte loads and one 16-byte store per cycle via a 32 KB 8-way set associative write-back L1 data cache.
- The floating point unit is capable of a single 256-bit AVX operation per cycle.

Zen Pipeline

- Can decode four instructions per cycle.
- Can deliver 6 operations per cycle to schedulers.
- load and store can perform two 16 byte loads and one 16 bytestore per cycle.
- 4 integer ALUs
- 2 AGUs (2R 1W)
- 4 FP pipelines

Zen Pipeline



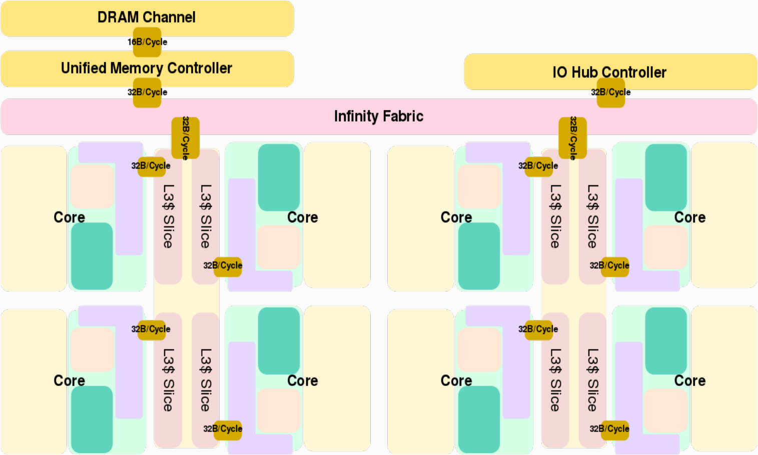
Clock Domains

- UMC Clock: The frequency at which Unified Memory Controller operates. The value is close to memory clock.
- Link Clock: The frequency at which Input and Output Hub controller operates with the chip.
- Fabric Clock: The frequency at which data fabric operates at and is similar to the memoryclock.
- Memory Clock: Internal and External memory clock.
- Core Clock: Frequency at which CPU Core and Cache operate.
- Uses clock gating (to save power)

Sockets and Connectivity

1. supports the AM4 socket, TR4 socket (threadgripper) and SP3 socket.
2. AM4 socket ensures DDR4, NVMe support.
3. The server processors use SP3 socket (8 channel DDR4).
4. SoC (System on chip) design.
5. USB, SATA and memory controllers are integrated on the same chip site.

System on Chip

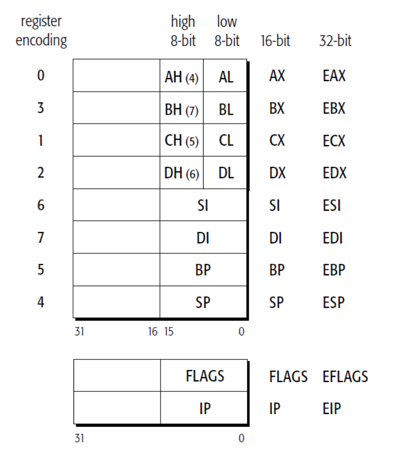


Instruction Set Architecture

General Purpose Programming - Registers i

The AMD 64 architecture has a total of 16, 64 bit general purpose registers - RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8, R9, R10, E11, R12, R13, R14 and R15. As compared to the x86 architecture, AMD64 has 8 new GPRs.

General Purpose Programming - Registers ii

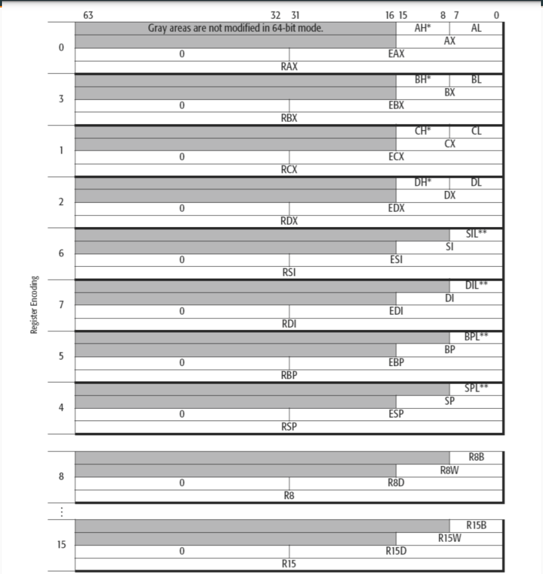


register encoding	high 8-bit	low 8-bit	16-bit	32-bit
0	AH (4)	AL	AX	EAX
3	BH (7)	BL	BX	EBX
1	CH (5)	CL	CX	ECX
2	DH (6)	DL	DX	EDX
6	SI		SI	ESI
7	DI		DI	EDI
5	BP		BP	EBP
4	SP		SP	ESP

		16-bit	32-bit
	FLAGS	FLAGS	EFLAGS
	IP	IP	EIP

Figure 3: General registers in legacy and compatibility modes

General Purpose Programming - Registers iii



*Not addressable when a REX prefix is used.

** Only addressable when a REX prefix is used.

Figure 4: GPRs in 64-Bit Mode

Data Conversion - Endian Interchange Byte Swap

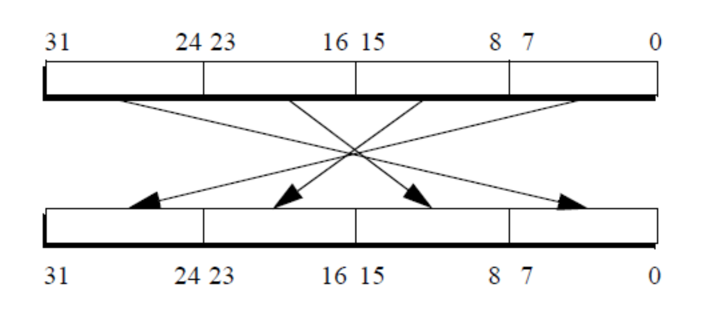
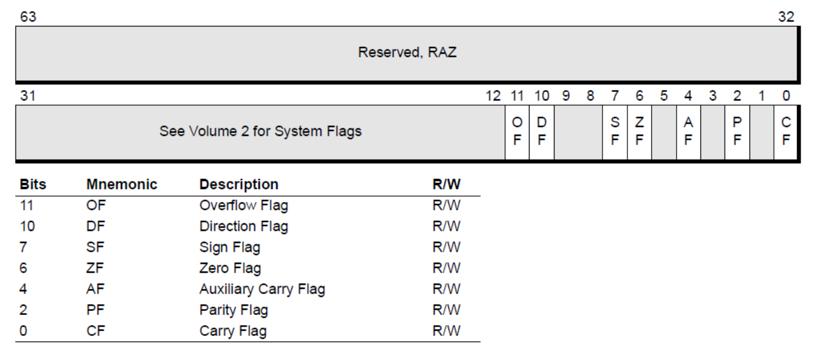


Figure 5: BSWAP Doubleword Exchange used in converting between Little Endian and Big Endian, Figure 3-8. [?]

Flag Registers i

1. A flag register is a special purpose register. Depending on the value of the result after any arithmetic and logical operation, the flag bits will either become 1 or 0 0 being the reset and 1 being the set value. If the MSB (Most Significant Bit) indicates 0, then the number is positive, and if it indicates 1, then it is negative.
2. An overflow flag can be indicated by the processor in case there is any arithmetic overflow

Flag Registers ii



63	32
Reserved, RAZ	

31	12	11	10	9	8	7	6	5	4	3	2	1	0
See Volume 2 for System Flags		OF	DF			SF	ZF		AF		PF		CF

Bits	Mnemonic	Description	R/W
11	OF	Overflow Flag	R/W
10	DF	Direction Flag	R/W
7	SF	Sign Flag	R/W
6	ZF	Zero Flag	R/W
4	AF	Auxiliary Carry Flag	R/W
2	PF	Parity Flag	R/W
0	CF	Carry Flag	R/W

Figure 6: Register Flags

Division

Division is the slowest of all integer arithmetic operations and should be avoided wherever possible. Possibly by replacing i/j/k with i/(j $\times$ k).

Preliminaries - Overflow

Unsigned (Signed) Decimal	Binary
0	0000 0000
1	0000 0001
⋮	⋮
64	0100 0000
⋮	⋮
127	0111 1111
128 (-128)	1000 0000
129 (-127)	1000 0001
⋮	⋮
192 (-64)	1100 0000
⋮	⋮
255 (-1)	1111 1111
256 (0)	$\boxed{1}$ 0000 0000

Table 1: Overflow in Binary Arithmetic

ASCII Adjust Instructions i

	43		0010 1011		0100 0011
ADD +	29	+	0001 1011	+	0010 1001
	72		0100 0110		0110 1100
				AAA	0111 0010

Table 2: An ASCII Adjust on an addition and division of a packed binary coded decimal

The **ASCII Adjust Instructions** AAA, AAD, AAM, AAS (ASCII Adjust after addition, division, multiplication and subtraction) apply corrections on non packed BCD values (i.e., when the BCD is stored in a byte register) such as passing the carry of one digit to the next digit. The **BCD** counterparts perform corrections (addition and subtraction only

ASCII Adjust Instructions ii

- DAA and DAS) on packed BCD values (the two nibbles in a byte represent two digits).

```
MOV AL, 0000 0111
MOV BH, 0000 1000
MUL BH ; Result is stored in AX
; AX = 0011 1000  = 45 DEC
AAM ; 0100 0101 = 45 BCD
```

AAM ASCII adjust after multiplication on a packed BCD

Increment and Decrement

The increment (INC) and decrement (DEC) instructions are identical to the ADD and SUB instructions except in that they do not affect the Carry flag (0 bit of the RFLAGS register).

Multiplication

The MUL instruction takes only one operand (multiplicand) which is a factor and the result is stored in the same register which is treated as an accumulator.

```
MOV AL, 12
MOV BX, 13
MUL BX; stores 156 in AX
```

Operand size	Multiplicand	Result
Byte	AL (8 bit)	AX (16 bit)
Word	AX	rDX and rAX (64 bits each)
Doubleword	AX	rDX and rAX
Quadword	AX	rDX and rAX

Table 3: The register reference used for the MUL instruction based on the operand size

Rotate and Shift i

The **Rotate (and Shift)** instructions perform cyclic (or non-cyclic) rotations of the operand by *count* number of bits. The last bit to be rotated out is stored in the *Carry Flag - CF*. In single left bit rotates, the *Overflow flag - OF* is set to *CF XOR (MSB of result)*. In single right bit rotates, the *Overflow flag - OF* is set to *2nd MSB of result XOR (MSB of result)*. Bit rotations are used in character conversion including cryptography techniques. SAR (shift Arithmetic Right) ignores (preserves) the MSB and flushes out the (2nd MSB) instead.

```
MOV AL, 1011 1101; (- 0100 0011 stored - 67)
SAR AL; 1101 1110 (- 0010 0010 = - 32 )
```

Stack Operation

1. POP (Pop word off stack), POPA/POPAD (Pop all registers onto stack), PUSH, PUSHA, PUSHAD commands are used to pop values and stack them onto the general purpose registers (GPRs).
2. rBP is used as the stack base pointer and rSP is used as the stack reference pointers.

Bit Manipulation

Bit manipulation instructions manipulate individual bits in a register for purposes such as controlling low-level devices, correcting algorithms, and detecting errors.

1. BEXTR (Bit Field extract) is used to extract a contiguous field of bits

SETcc - Conditional SET

The conditional set instructions set the byte operand to 0 or 1 depending on the condition (by reading the RFLAGS bits) based on conditions as can be found in Table 3-5 in [**?**].

Code Sample 1 i

```
cmp ecx, 5 ; test if ecx equals 5
jnz Continue ; test condition and skip if not met
mov eax, ebx ; move
Continue: ; continuation

cmp ecx, 5 ; test if ecx equals to 5
cmovz eax, ebx ; test condition and move
```

This example elucidates use of the compare statement in conjunction with the RFLAGS register and of the conditional move (CMOVcc) statements. The advantage of CMOVcc statements, apart from cleaner code is it avoids branch prediction penalties caused by conditional jumps.

Cache and Memory Management

The (L/S/M)FENCE (load/store/memory) instructions force orders on memory access.
All memory loads (stores) preceding the LFENCE (SFENCE) (in program order) are completed prior to completing memory loads following the LFENCE (SFENCE). Memory loads cannot be reordered around an LFENCE (SFENCE) instruction, but other non-serializing instructions (such as memory writes (reads)) can be reordered around the LFENCE(SFENCE).
All memory accessess (read and write) preceding the MFENCE (in program order) are completed prior to completing any memory access following the MFENCE. Memory accesses cannot be reordered around an MFENCE instruction. In AMD64, MFENCE is a serialising instruction.

Serialising Instructions

Serialising Instructions (MFENCE, CPUID, IRET) alter the software-visible state. They force the processor to commit the serializing instruction and all previous instructions, then restart instruction fetching at the next instruction (flushes speculatively fetched instructions [1]). A *locked instruction* (contains the LOCK instruction prefix) is used to perform an atomic read-modify-write operation on a memory operand. It needs exclusive access to the memory location for the duration of the operation.

I/O Instructions - *IN* Instructions are not executed until all previous stores to memory and I/O address space are complete. Instructions following an *OUT* instruction are not executed till executed until all previous stores to memory and I/O-address space are complete, including the store performed by the OUT.

[1] Speculative fetching, refer to branch miss penalty in conditional set

SSE Instructions

The Streaming SIMD (Single Instruction Multiple Data) instructions speed up common operations by passing multiple byte data into a single 128-bit XMM or 256-bit YMM register and executing a single instruction on these. An example is the dot product (VPMADDWD - Vector Packed Multiply Words and Add Doublewords) shown on the next slide.

Code Sample 2 i

This computes the dot product of 8 2-vectors with X coordinates in YMM0 and Y coordinates in YMM1 and stores in the destination (first) operand.

```
MOV YMM0, 0x11112222333344445555666677778888
MOV YMM1, 0x11112222333344445555666677778888
VPMADDWD YMM0, YMM1
```

Code Sample 2 ii

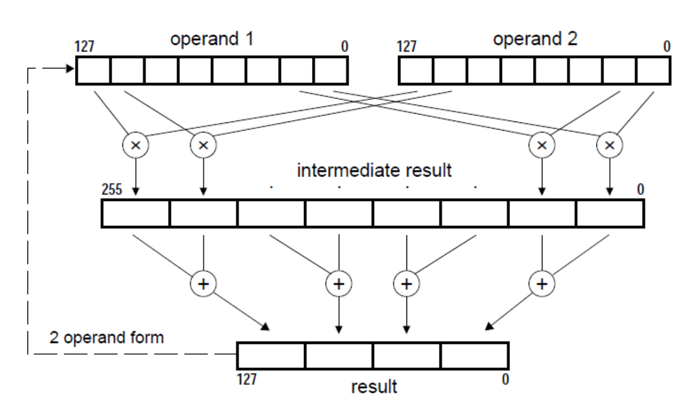


Figure 7: (V)PMADDWD Multiply-Add Operation (Figure 4-35. [?])

Code Sample 2 iii

The following code multiplies 8 32-bit integers of the 8-tuple YMM0 with the corresponding entries of the YMM1.

```
MOV YMM0, 0x11112222333344445555666677778888
MOV YMM1, 0x11112222333344445555666677778888
VPMULLD YMM0, YMM1
```

Applications

Specific Applications

1. SSE Instruction Set provides the benefit of running instructions across multiple elements in parallel
2. Music synthesis, speech synthesis, speech recognition, audio and video compression (encoding) and decompression (decoding), 2D and 3D graphics, streaming video (up to high-definition TV), and digital signal processing (DSP)
3. Dense systems of linear equations, including matrix and vector-space operations. Simulations

Other Applications

1. Popularly used in the domain of gaming
2. Have four series : Ryzen 3, Ryzen 5, Ryzen 7 and Ryzen 9
3. Versatile for the consumers