Wei Fei (2538810)
EECS 665 Lab8 Report
Elluru, Bharath Chandra
11/12/15

I will choose X86-64 and ARM to do some research.

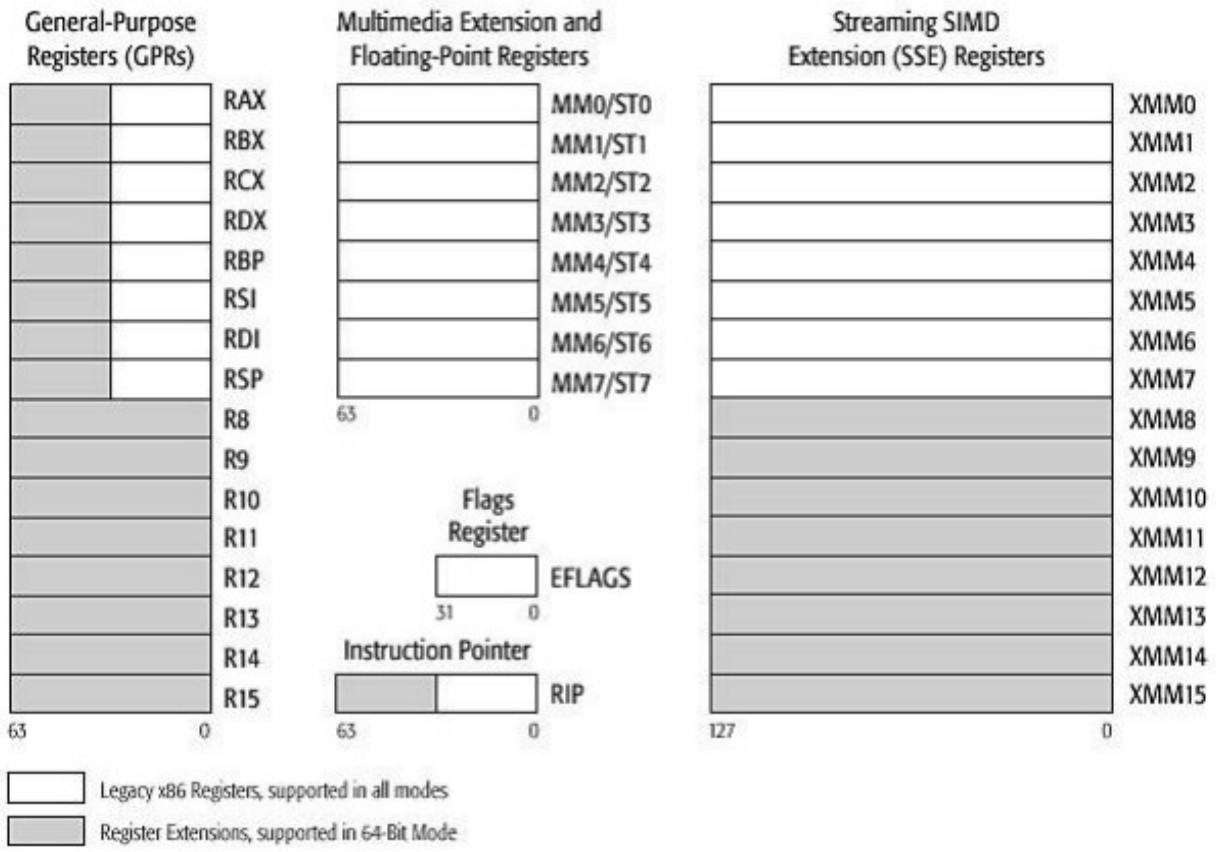
1. Basic Information

Who developed the processor? Is the processor a RISC or a CISC architecture? What is the bit-width of processor instructions? What does the memory architecture of the processor look like? How are instructions encoded by the processor?

X86-64:

AMD implementedm, created X86-64 and released in 2000. The processor is RISC architecture with 64-bit instruction set.

The memory architecture is like this: [1]

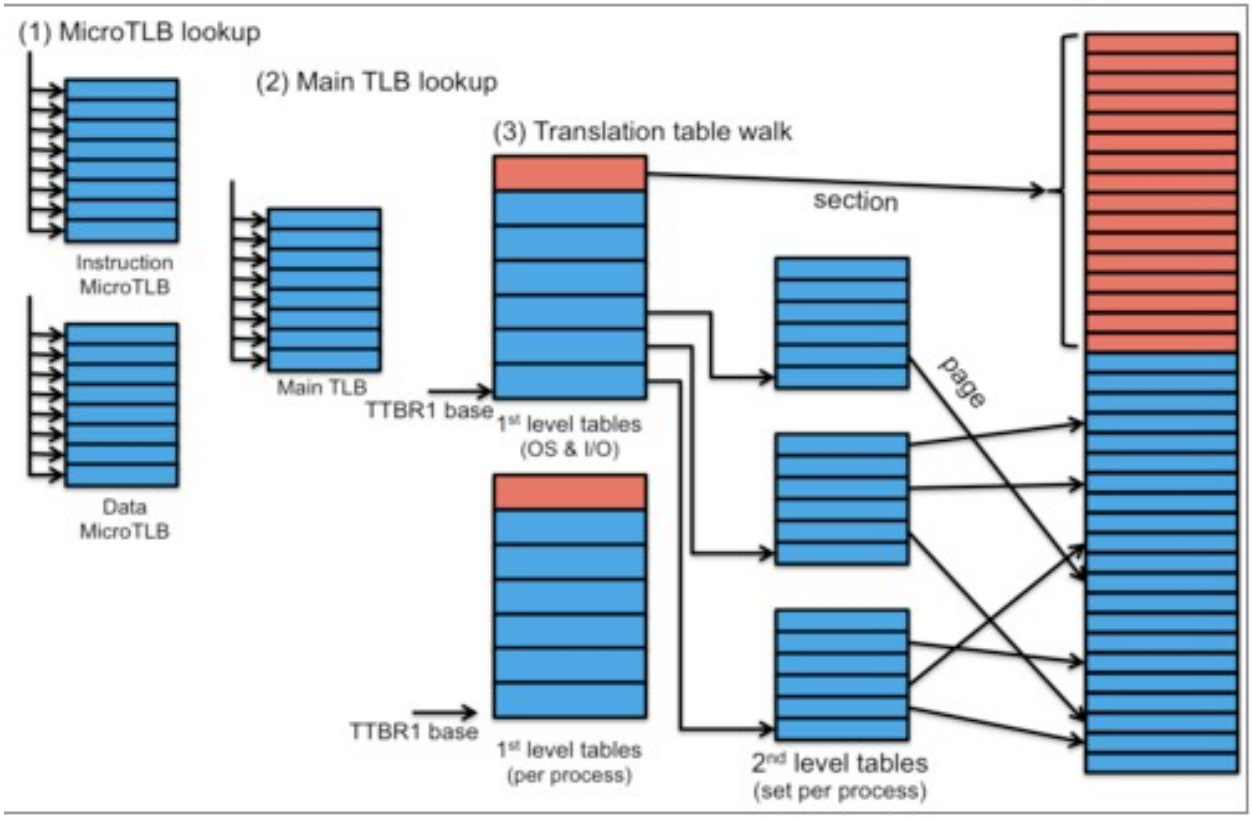


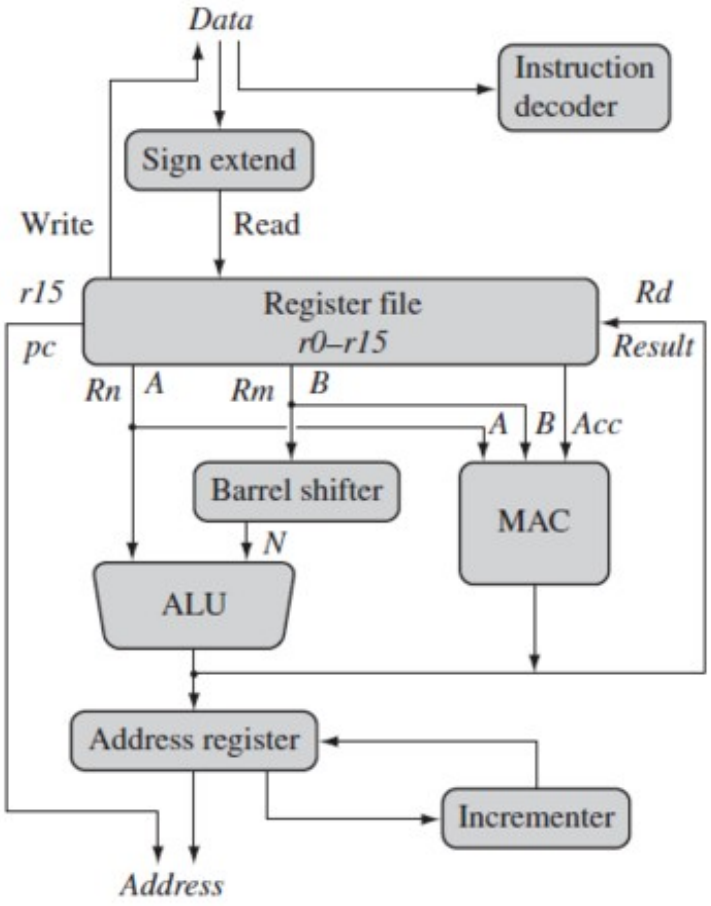
The instruction immediate value is encoded these using Ivds code. I code means Immediate, v means word or doubleword (imm16 or imm32). The most important part is ds code, which means doubleword, sign-extended to 64 bits for 64-bit operand size. The instruction has no ModR/M byte; the address of the operand is encoded in the instruction; no base register, index register, or scaling factor can be applied. [2]

ARM:

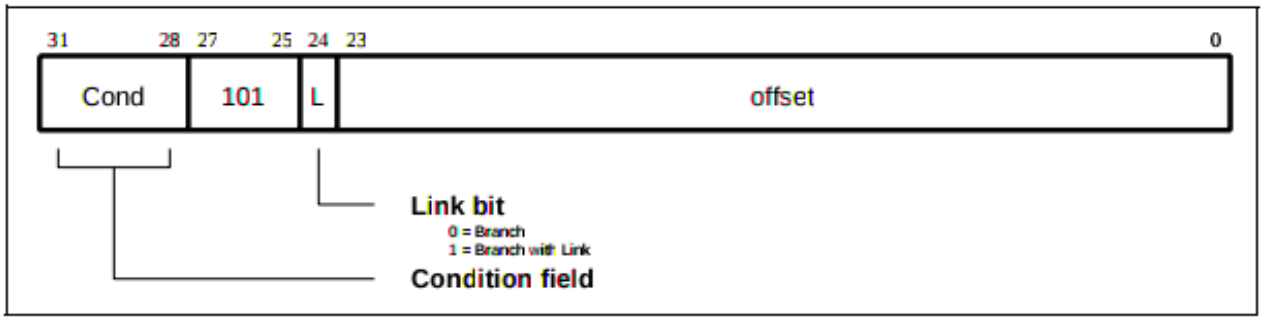
The ARM processor was designed and developed by British company ARM holdings. It belongs to RISC architecture. The ARM can support both of 32-bit and 64-bit instruction set.

The memory architecture is like this [3][4]:





One of the instruction encoding for ARM is called branch insructions [5].



Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

2. Supported Data Types

What data types are supported? What are the bit-widths of those data types? Are the data types stored in big-endian, little-endian, or some other format?

X86-64:

X86-64 can support a lot of data types [6]:

C declaration	Intel data type	GAS suffix	x86-64 Size (Bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
unsigned	Double word	l	4
long int	Quad word	q	8
unsigned long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	d	8
long double	Extended precision	t	16

Bit-width	Data type
8-bit	char
16-bit	short
32-bit	int, unsigned, float
64-bit	long int, unsigned long, char *, double
128-bit	long double

The data types stored in little-endian format.

ARM:

Data types and bit-widths of ARM [7]:

Type	Size in bits	Natural alignment in bytes
`char`	8	1 (byte-aligned)
`short`	16	2 (halfword-aligned)
`int`	32	4 (word-aligned)
`long`	32	4 (word-aligned)
`long long`	64	8 (doubleword-aligned)
`float`	32	4 (word-aligned)
`double`	64	8 (doubleword-aligned)
`long double`	64	8 (doubleword-aligned)
All pointers	32	4 (word-aligned)
`bool` (C++ only)	8	1 (byte-aligned)
`_Bool` (C only[a])	8	1 (byte-aligned)
`wchar_t` (C++ only)	16	2 (halfword-aligned)

Integers are represented in two's complement form. The low word of a long long is at the low address in little-endian mode, and at the high address in big-endian mode. For double and long double quantities the word containing the sign, the exponent, and the most significant part of the mantissa is stored with the lower machine address in big-endian mode and at the higher address in little-endian mode.

3. Register Set

What registers are in the machine? What is the bit-width of each register? What kind of register is each register?

X86-64:

The registers information for X86-64 is listed here[5]:

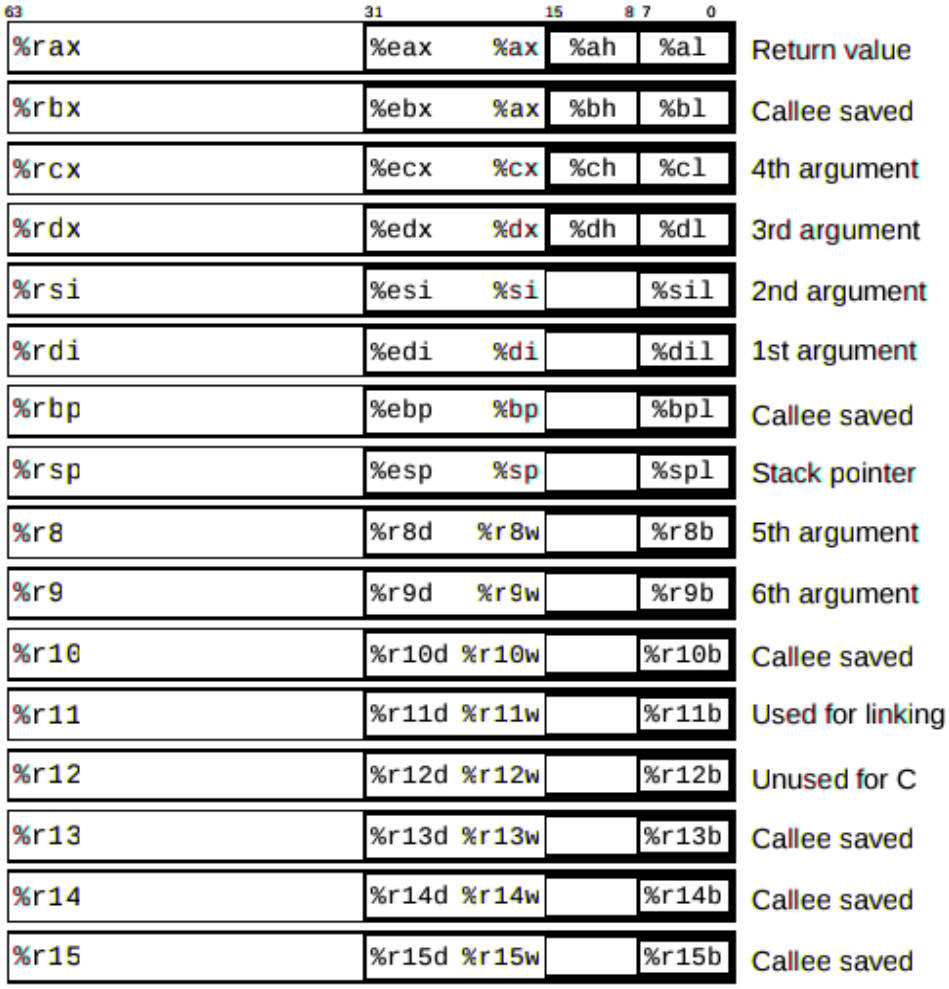


Figure 2: **Integer registers.** The existing eight registers are extended to 64-bit versions, and eight new registers are added. Each register can be accessed as either 8 bits (byte), 16 bits (word), 32 bits (double word), or 64 bits (quad word).

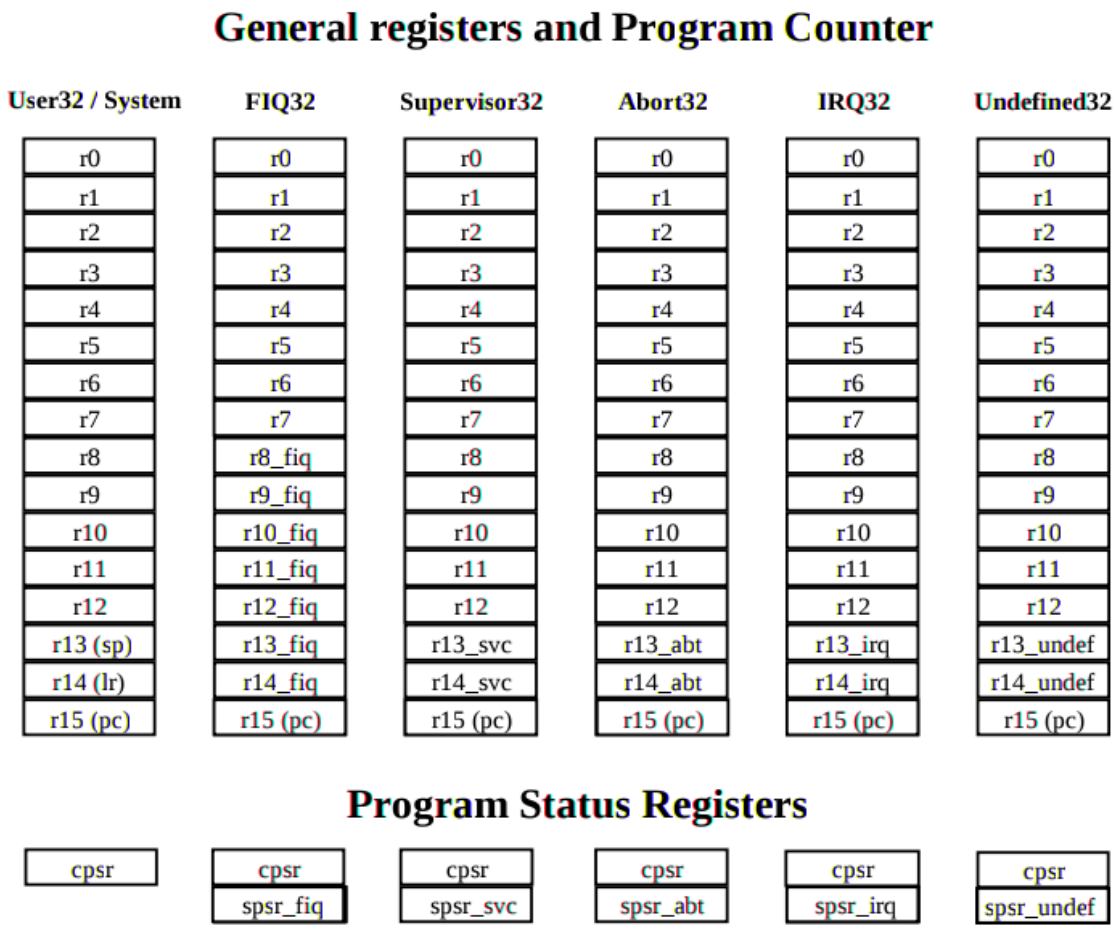
All registers are 64 bits long. The 64-bit extensions of the IA32 registers are named %rax, %rcx, %rdx, %rbx, %rsi, %rdi, %rsp, and %rbp. The new registers are named %r8–%r15. All registers are 64 bits long. The 64-bit extensions of the IA32 registers are named %rax, %rcx, %rdx, %rbx, %rsi, %rdi, %rsp, and %rbp. The new registers are named %r8–%r15. The low-order 32 bits of each register can be accessed directly. This gives us the familiar registers from IA32: %eax, %ecx, %edx, %ebx, %esi,

%edi, %esp, and %ebp, as well as eight new 32-bit registers: %r8d–%r15d. The low-order 16 bits of each register can be accessed directly, as is the case for IA32. The word-size versions of the new registers are named %r8w–%r15w. The low-order 8 bits of each register can be accessed directly. This is true in IA32 only for the first 4 registers (%al, %cl, %dl, %bl). The byte-size versions of the other IA32 registers are named %sil, %dil, %spl, and %bpl. The byte-size versions of the new registers are named %r8b–%r15b. For backward compatibility, the second byte of registers %rax, %rcx, %rdx, and %rbx can be directly accessed by instructions having single-byte operands.

ARM:

ARM has 37 registers in total, all of which are 32-bits long. 1 dedicated program counter, 1 dedicated current program status register, 5 dedicated saved program status registers, and 30 general purpose registers[6].

Here is the register Organisation graph:



4. Describe the instruction encodings for one Arithmetic instruction (Like Add, Mul), one Memory Access instruction (Like load, store), one Call instruction and one Conditional branch instruction?

X86-64:

- Arithmetic instruction [8]:

Immediate: the value is stored in the instruction. ADD EAX, 14 ; add 14 into 32-bit EAX

- Memory Access instruction [9]:

Instruction Encoding

Mnemonic	Opcode	Description
MOVBE *reg16*, *mem16*	0F 38 F0 */r*	Load the low word of a general-purpose register from a 16-bit memory location while swapping the bytes.
MOVBE *reg32*, *mem32*	0F 38 F0 */r*	Load the low doubleword of a general-purpose register from a 32-bit memory location while swapping the bytes.
MOVBE *reg64*, *mem64*	0F 38 F0 */r*	Load a 64-bit register from a 64-bit memory location while swapping the bytes.
MOVBE *mem16*, *reg16*	0F 38 F1 */r*	Store the low word of a general-purpose register to a 16-bit memory location while swapping the bytes.
MOVBE *mem32*, *reg32*	0F 38 F1 */r*	Store the low doubleword of a general-purpose register to a 32-bit memory location while swapping the bytes.
MOVBE *mem64*, *reg64*	0F 38 F1 */r*	Store the contents of a 64-bit general-purpose register to a 64-bit memory location while swapping the bytes.

- Call instruction [9];

JMP *rel8off*	EB *cb*	Short jump with the target specified by an 8-bit signed displacement.
JMP *rel16off*	E9 *cw*	Near jump with the target specified by a 16-bit signed displacement.
JMP *rel32off*	E9 *cd*	Near jump with the target specified by a 32-bit signed displacement.

- Conditional branch instruction

CMOVO *reg16*, *reg/mem16* CMOVO *reg32*, *reg/mem32* CMOVO *reg64*, *reg/mem64*	0F 40 */r*	Move if overflow (OF = 1).
CMOVNO *reg16*, *reg/mem16* CMOVNO *reg32*, *reg/mem32* CMOVNO *reg64*, *reg/mem64*	0F 41 */r*	Move if not overflow (OF = 0).

ARM:

- Arithmetic instruction [8]:

UMULL R1,R4,R2,R3 ; R4,R1:=R2*R3

UMLALS R1,R5,R2,R3 ; R5,R1:=R2*R3+R5,R1

; condition codes

- Memory Access instruction

<LDR| STR>{cond}{B}{T}Rd,<Address>

LDR - load from memory into a register

STR - store from a register into memory

```
LDR    R0,[R1],R1
```

<LDR| STR>Rd,[Rn],{+/-}Rn{,<shift>}

- Call instruction

```
BAL       here      ; assembles to 0xEAFFFFFE (note effect of PC offset)
B         there     ; ALways condition used as default

CMP       R1,#0     ; compare R1 with zero and branch to fred if R1
BEQ       fred      ; was zero otherwise continue to next instruction

BL        sub+ROM   ; call subroutine at computed address

ADDS      R1,#1     ; add 1 to register 1, setting CPSR flags on the
BLCC      sub       ; result then call subroutine if the C flag is clear,
                    ; which will be the case unless R1 held 0xFFFFFFFF
```

- Conditional branch instruction [6];

BAL here ; assembles to 0xEAFFFFFE (note effect of

; PC offset).

B there ; Always condition used as default.

CMP R1,#0 ; Compare R1 with zero and branch to fred

; if R1 was zero, otherwise continue

BEQ fred ; continue to next instruction.

5. Is the instruction set zero-address, one-address, two-address, three-address or something else. Justify your answer?

X86-64:

It uses two-address code and three-address code.

ARM:

It used two-address code

References:

[1]: Advanced Micro Devices, Inc. x86-64TM Technology White Paper
http://people.cs.clemson.edu/~mark/464/x86-64_wp.pdf

[2]: X86 Opcode and Instruction Reference http://ref.x86asm.net/

[3]: Memory Management: Paging https://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html

[4]: ARM Microprocessors
http://www.mouloudrahmani.com/ElectricalEngineering/Embedded/MicroARM

[5]: x86-64 Machine-Level Programming
http://www.cs.cmu.edu/~fp/courses/15213-s07/misc/asm64-handout.pdf

[6]: The ARM Instruction Set http://simplemachines.it/doc/arm_inst.pdf

[7]: ARM The Architecture for the Digital World
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0491c/Babfcgfc.html

[8]: Introduction to x64 Assembly
https://software.intel.com/en-us/articles/introduction-to-x64-assembly

[9]: AMD64 Architecture Programmer's Manual by AMD64 Technology
http://support.amd.com/techdocs/24594.pdf