CSCE313 Midterm Cheat Sheet

---

1. x86-64 Registers

Register	Usage
%rax	Return value / temporary
%rbx	Callee-saved general-purpose
%rcx	4th argument to functions (caller-saved)
%rdx	3rd argument to functions (caller-saved)
%rsi	2nd argument to functions (caller-saved)
%rdi	1st argument to functions (caller-saved)
%rbp	Base/frame pointer (callee-saved)
%rsp	Stack pointer (special-purpose, always points to top of stack)
%r8 – %r15	Additional arguments / general-purpose
%rip	Instruction pointer (special-purpose)

Caller-saved (volatile): %rax, %rdi, %rsi, %rdx, %rcx, %r8-%r11
Callee-saved (non-volatile): %rbx, %rbp, %r12-%r15

---

2. Function Call Conventions (System V AMD64)

- Arguments 1–6: %rdi, %rsi, %rdx, %rcx, %r8, %r9
- Return value: %rax
- Caller saves volatile registers if it needs them after the call
- Callee saves non-volatile registers it uses

Stack Frame Setup Example:

```
pushq %rbp # save old frame pointer
movq %rsp, %rbp # set new frame pointer
subq $X, %rsp # allocate local variables
...
leave
ret
```

---

3. Processes & fork/exec

fork():

- Creates a **new process** (child) with a **copy of memory**
- Child gets its **own PID**
- Returns 0 in child, child PID in parent

exec family (execl, execvp, execlp, etc.):

- Replaces the **current process image** with a **new program**
- Does not create a new process

posix_spawn():

- Combines `fork` + `exec` efficiently
- Uses fewer resources than `fork` for creating a new process

wait()/waitpid():

- Parent blocks until child exits
- Can retrieve **exit status**

PID allocation example:

- Parent PID = 1000 → first child PID = 1001 → grandchild PID = 1002

---

4. Pipes & IPC

Type	Lifetime	Example Usage
Unnamed pipe	Exists only while processes are running	pipe(fd) between parent & child
Named pipe (FIFO)	Exists in filesystem	mkfifo mypipe; open("mypipe", ...)

Pipe Usage in C:

```
int fd[2]; // fd[0] = read end, fd[1] = write end
pipe(fd);
write(fd[1], buffer, size);
read(fd[0], buffer, size);
```

Shell Example:

```
ls -l | tr a-z A-Z
```

- Output of `ls -l` → input of `tr`

---

5. Unix File I/O

Call	Type	Notes
open()	System call	returns **file descriptor**
close()	System call	closes file descriptor
read(fd, buf, n)	System call	unbuffered
write(fd, buf, n)	System call	unbuffered
fopen() / fclose()	Library	buffered I/O
fread() / fwrite()	Library	buffered I/O
dup()/dup2()	System call	duplicate file descriptors

Standard File Descriptors:

- 0 → stdin
- 1 → stdout
- 2 → stderr

Redirection Example:

```
ls > output.txt
```

- Shell opens `output.txt` and **redirects stdout** to the file

---

6. GDB Quick Reference

Command	Usage
break <line/function>	Set breakpoint
run	Start program
next	Step over a function call
step	Step into a function call
continue	Continue until next breakpoint
print	Print variable value
bt	Backtrace (stack trace)
info locals	List all local variables in current frame
info registers	Show register values

Tips:

- `next` = go to next line **without entering functions**

- `step` = go to next instruction **including function calls**
- Uninitialized local variables may show garbage

---

7. Common Commands

Linux:

- ls, cd, pwd → navigation
- cat file, less file → view file
- chmod, chown → permissions

Git:

- git status → check changes
- git add → stage changes
- git commit -m "msg" → commit changes
- git push → upload to remote
- git pull → fetch + merge

Compiling with Debug Info:

```
gcc -g file.c -o program
```

Common x86-64 Instructions Cheat Sheet

Instruction	Syntax Example	Description	Notes / ARM Comparison
mov	movq %rax, %rbx	Copy value from source to destination	ARM: MOV Rd, Rn
add	addq %rsi, %rdi	Add source to destination, store in destination	ARM: ADD Rd, Rn, Rm
sub	subq %rdx, %rax	Subtract source from destination, store in destination	ARM: SUB Rd, Rn, Rm
imul	imulq %rbx, %rax	Signed multiply	ARM: MUL Rd, Rn, Rm
idiv	idivq %rbx	Signed divide: %rax / operand, quotient → %rax, remainder → %rdx	ARM: SDIV Rd, Rn, Rm
lea	leaq 8(%rbp,%rcx,4), %rax	Load effective address (pointer arithmetic)	ARM: ADD Rd, Rn, Rm, LSL #2 for scaled indexing

Instruction	Syntax Example	Description	Notes / ARM Comparison
push	pushq %rbp	Push value onto stack	ARM: PUSH {Rn}
pop	popq %rbp	Pop value from stack	ARM: POP {Rn}
call	call foo	Call function (push return address)	ARM: BL label
ret	ret	Return from function	ARM: BX LR
cmp	cmpq %rax, %rbx	Compare two values (sets flags)	ARM: CMP Rn, Rm
jmp	jmp label	Unconditional jump	ARM: B label
je / jne	je label	Jump if equal / not equal (conditional)	ARM: BEQ label / BNE label
test	testq %rax, %rax	Bitwise AND, sets flags	ARM: TST Rn, Rm
and	andq %rax, %rbx	Bitwise AND	ARM: AND Rd, Rn, Rm
or	orq %rax, %rbx	Bitwise OR	ARM: ORR Rd, Rn, Rm
xor	xorq %rax, %rbx	Bitwise XOR	ARM: EOR Rd, Rn, Rm
inc	incq %rax	Increment by 1	ARM: ADD Rd, Rn, #1
dec	decq %rax	Decrement by 1	ARM: SUB Rd, Rn, #1
nop	nop	No operation	ARM: NOP
movzbq	movzbq %al, %rax	Move byte → quadword with zero extension	ARM: UXTB Rd, Rn

Notes:

- `q` suffix in x86-64 instructions = quadword (64-bit)
- `l` suffix = long (32-bit), `w` = word (16-bit), `b` = byte (8-bit)
- Conditional jumps rely on flags set by `cmp` or `test` instructions
- `lea` is frequently used for pointer arithmetic without changing memory

---

x86-64 Conditional Jump Cheat Sheet

Instruction	Jump Condition	Flags Used / Meaning
je / jz	Jump if equal / zero	ZF = 1 (Zero Flag set)
jne / jnz	Jump if not equal / not zero	ZF = 0

Instruction	Jump Condition	Flags Used / Meaning
ja / jnbe	Jump if above (unsigned)	CF = 0 and ZF = 0
jae / jnb	Jump if above or equal (unsigned)	CF = 0
jb / jnae	Jump if below (unsigned)	CF = 1
jbe / jna	Jump if below or equal (unsigned)	CF = 1 or ZF = 1
jg / jnle	Jump if greater (signed)	ZF = 0 and SF = OF
jge / jnl	Jump if greater or equal (signed)	SF = OF
jl / jnge	Jump if less (signed)	SF ≠ OF
jle / jng	Jump if less or equal (signed)	ZF = 1 or SF ≠ OF
jo	Jump if overflow	OF = 1
jno	Jump if not overflow	OF = 0
js	Jump if sign (negative)	SF = 1
jns	Jump if not sign (positive)	SF = 0
jp / jpe	Jump if parity even	PF = 1
jnp / jpo	Jump if parity odd	PF = 0

Notes:

- ZF = Zero Flag → set when result = 0
- CF = Carry Flag → set on unsigned overflow / borrow
- SF = Sign Flag → set if result is negative
- OF = Overflow Flag → set on signed overflow
- PF = Parity Flag → set if low byte has even number of 1 bits

---