

# SUS-32 “Among Us” Reference Sheet ☹

## 1. Overview

- a. 32-bit word, 32-bit instructions
- b. Themed after Among Us!!!!
- c. 3 formats
  - i. C-type (Crew): register-register (like R-type)
  - ii. T-type (Task): immediates / loads / stores / branches (like I-type)
  - iii. V-type (Vent): jumps / calls (like J-type)
- d. Extra flags:
  - i. SF = Sus Flag (1 bit, set by REPORT, CHECKSUS, FAKETASK, etc.)
  - ii. PBASE = base address of player table (conceptual constant)\
  - iii. PC = Program Counter

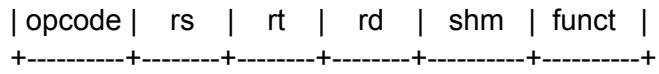
## 2. Registers (Crew Register File)

Num	Name	Role	Saved? Y/N
0	\$void	Constant 0	N/A
1	\$host	Assembler temporary	No
2 - 3	\$report0 - 1	Function results / expression eval	No
4 - 7	\$crewA - D	Arguments	No
8 - 15	\$task 0 - 7	Temporaries	No
16 - 23	\$safe0 - 7	Saved temporaries	Yes
24 - 25	\$task8 - 9	Extra temporaries	No
26 - 27	\$kernel0 - 1	Reserved for OS	No
28	\$map	Global pointer / game state base	Yes
29	\$stack	Stack pointer	Yes
30	\$frame	Frame pointer	Yes
31	\$return	Return address	No

## 3. Instructions Formats

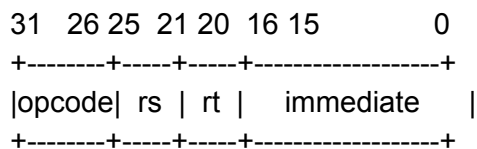
### 3.1 C-type (Crew, like R-type)

31    26 25   21 20    16 15    11    10   6 5   0  
+-----+-----+-----+-----+-----+-----+



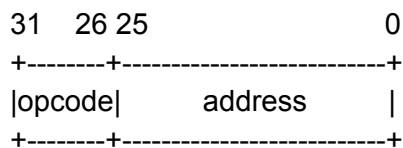
Usually opcode = 000000<sub>2</sub> (0x00)  
 Operation chosen by funct

### 3.2 T-type (Task, like I-type)



imm usually sign-extended  
 Branch target: PC = PC+4 + (SignExt(imm) << 2)

### 3.3 V-type (Vent, like J-type)



Jump target: PC = { PC[31:28], address, 2'b00 }

## 4. Core Instructions (Basic Tasks)

### 4.1 C-type Core (opcode = 0x00)

Mnemonic	Syntax	Effect	funct
TASKADD	TASKADD rd, rs, rt	$R[rd] = R[rs] + R[rt]$	0x20
TASKSUB	TASKSUB rd, rs, rt	$R[rd] = R[rs] - R[rt]$	0x22
TASKAND	TASKAND rd, rs, rt	$R[rd] = R[rs] \& R[rt]$	0x24
TASKOR	TASKOR rd, rs, rt	$R[rd] = R[rs]   R[rt]$	0x25
TASKXOR	TASKXOR rd, rs, rt	$R[rd] = R[rs] \wedge R[rt]$	0x26

### 4.2 T-type Core

Mnemonic	Syntax	Effect	opcode
----------	--------	--------	--------

TASKSET	TASKSET rt, imm	R[rt] SignExt(imm)	0x08
LOADTASK	LOADTASK rt, off(rs)	$R[rt] = M[R[rs] + \text{SignExt(off)}]$	0x23
SAVETASK	SAVETASK rt, off(rs)	$M[R[rs] + \text{SignExt(off)}] = R[rt]$	0x2B
SUSBEQ	SUSBEQ rs, rt, label	if $R[rs] == R[rt]$ branch	0x04
SUSBNE	SUSBNE rs, rt, label	if $R[rs] != R[rt]$ branch	0x05

### 4.3 V-type Core

Mnemonic	Syntax	Effect	opcode
VENT	VENT label	$PC = \{PC[31:28], \text{address}, 2'b00\}$	0x02
VENTLINK	VENTLINK label	$\$return = PC+8; PC = \{PC[31:28], \text{address}, 2'b00\}$	0x03

## 5. SUS Special Instructions

### 5.1 C-type SUS (opcode = 0x00)

Mnemonic	Syntax	Effect	funct
REPORT	REPORT rs	$SF = (R[rs] != 0)$	0x28
SABOTAGE	SABOTAGE rd, rs, rt	$R[rd] = R[rs] \wedge R[rt]$	0x29
FAKETASK	FAKETASK rd, rs	$R[rd] = R[rs]; SF = 1$	0x2A
CLEARVOTES	CLEARVOTES	clears predefined "vote" registers	0x2B
CHECKSUS	CHECKSUS rs, rt	$SF = (R[rs] == R[rt]) ? 0 : 1$	0x2C
RANDOMTASK	RANDOMTASK rd	$R[rd] = \text{PRNG\_next}()$	0x2D

Encoding notes for single-operand/no-operand C-type:

- REPORT: rt=rd=0, shamt=0
- FAKETASK: rt=0, shamt=0
- CLEARVOTES, RANDOMTASK: rs=rt=0, shamt=0

## 5.2 T-type SUS (custom opcodes)

Mnemonic	Syntax	Effect	opcode
VENTIFNEG	VENTIFNEG rs, label	if R[rs]<0 then branch	0x10
VOTEOUT	VOTEOUT rs	M[PBASE + 4*R[rs]] = 0 (eject player)	0x12
SCANMED	SCANMED rt	R[rt] = M[0xFFFF0000] (medbay status)	0x13

## 5.3 V-type SUS

Mnemonic	Syntax	Effect	opcode
EMERGENCY	EMERGENCY label	\$return = PC+4; PC = meeting address	0x11

## 6. Key Encoding Notes

- Sign extension:  $\text{SignExt}(\text{imm}) = \{16\{\text{imm}[15]\}, \text{imm}[15:0]\}$
- Branch address:  $\text{BranchAddr} = \text{SignExt}(\text{imm}) \ll 2$
- Vent/jump target: PC[31:28] (upper bits) + address (26 bits) + 00 (2 bits)
- Sus Flag (SF):
  - 0 = not sus / stories match
  - 1 = sus / mismatch / fake task

## SUS-32 Binary Encoding reference ☺

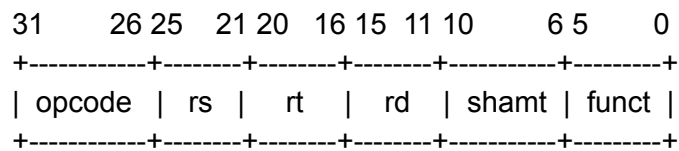
### 1. General Rules

- All instructions are 32 bits
- Bit numbering:
  - Bit 31 = most significant bit
  - Bit 0 = least significant bit
- Fields are interpreted differently depending on the instruction format
- All unused fields must be set to 0

### 2. C-Type (Crew / Register Format)

- Used for:
  - Arithmetic (TASKADD, TASKSUB)
  - Logic (TASKAND, TASKOR)
  - SUS special register instructions (REPORT, CHECKSUS, etc.)

## 2.1 C-Type Layout



## 2.2 C-Type Field Descriptions

Field	Bits	Description
opcode	31 - 26	Always 000000 <sub>2</sub> (0x00) for all C-type
rs	25 - 21	First source register
rt	20 - 16	Second source register
rd	15 - 11	Destination register
shamt	10 - 6	Shift amount (used only for shifts, else 0)
funct	5 - 0	Selects the exact operation

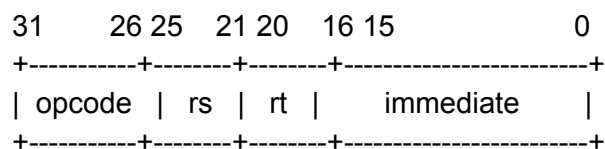
## 2.3 C-Type Encoding Notes

- Non-shift instructions:
  - Unused register fields must be 0
- Single-operand instructions (REPORT, FAKETASK, etc.):
  - shamt = 00000<sub>2</sub>
- No-operand instructions (CLEARVOTES, RANDOMTASK):
  - rs = rt = rd = shamt = 0

## 3. T-Type (Task / Immediate Format)

- Used for:
  - Loads and stores
  - Branches
  - Conditional venting
  - Medbay and vote instructions

### 3.1 T-Type Layout



### 3.2 T-Type Field Descriptions

Field	Bits	Description
opcode	31 - 26	Operation selector
rs	25 - 21	Base register / source register
rt	20 - 16	Destination register or comparison register
immediate	15 - 0	Constant, offset, or branch displacement

### 3.3 Immediate Handling

- Sign Extension (most T-type):
  - $\text{SignExt}(\text{imm}) = \{ 16\{\text{imm}[15]\}, \text{imm}[15:0] \}$
- Used by:
  - LOADTASK
  - SAVETASK
  - SUSBEQ
  - SUSBNE
  - VENTIFNEG

### 3.4 Branch Address Calculation

- For all branch-style T-type instructions:
  - $\text{BranchAddr} = \text{SignExt}(\text{immediate}) \ll 2$
  - $\text{PC} = \text{PC} + 4 + \text{BranchAddr}$

This keeps branches word-aligned.

### 3.5 T-Type Encoding Notes

- Instructions with unused fields:
  - VOTEOUT:  $\text{rt} = 0, \text{imm} = 0$
  - SCANMED:  $\text{rs} = 0, \text{imm} = 0$
- Immediate field may be reserved in future versions

## 4. V-Type (Vent / Jump Format)

- Used for:
  - VENT
  - VENTLINK
  - EMERGENCY

### 4.1 V-Type Layout



## 4.2 V-Type Field Descriptions

Field	Bits	Description
opcode	31 - 26	Jump operation
address	25 - 0	Target instruction address

## 4.3 Jump Target Construction

- PC = { PC[31:28], address, 2'b00 }

Top 4 bits come from PC + 4

Bottom 2 bits are always 00 (word alignment)

# 5. SUS-Specific Architectural Fields

## 5.1 Sus Flag (SF)

- 1-bit implicit flag
- Not stored in a register
- Modified by certain instructions

Instruction	SF Result
REPORT	SF = (R[rs] != 0)
FAKETASK	SF = 1
CHECKSUS	SF = (R[rs] == R[rt]) ? 0 : 1

## 5.2 Player Table Base (PBASE)

- Conceptual constant address
- Used by:
  - VOTEOUT
  - Not encoded directly in instructions
  - Defined by runtime / simulator

# 6. Summary by Format

Format	Uses	Key Control Field
C-Type	Arithmetic, logic, SUS register ops	funct
T-Type	Loads, stores, branches, I/O	opcode
V-Type	Vents and emergency jumps	opcode

## Example Programs

### **Program 1 - Compare Two Crew Reports (If Equal or Sus)**

Goal: Check if two values match. If they match, mark not sus. If they don't, mark sus.

Code:

```
# Assume:
# $task0 = first report value
# $task1 = second report value

main:
CHECKSUS $task0, $task1    # Set SF = 0 if equal, 1 if different
SUSBEQ  $task0, $task1, not_sus # If values equal, branch to not_sus

# Values are different → sus path
REPORT  $task0            # Nonzero → SF = 1 (someone is sus)
VENT    done              # Jump to end

not_sus:
REPORT  $void              # Report 0 → SF = 0 (no sus)
done:
VENT    done              # Stay here (infinite loop / end)
```

#### **What this shows:**

- CHECKSUS: compares two registers and updates the Sus Flag (SF).
- SUSBEQ: uses normal register equality to branch if they match.
- REPORT: shows how SF can also be set based on a value (0 vs nonzero).
- VENT: unconditional jump, used as a simple end marker.

### **Program 2 – Medbay Scan and Vote Out a Sus Player**

Goal: Read medbay status. If player is not confirmed crew, eject them with VOTEOUT.

Code:

```
# Assume:
# $crewA = player index (0..N-1)
# medbay word at 0xFFFF0000:
#   bit 0 = 1 → confirmed crew
#   bit 0 = 0 → not confirmed (possibly sus)
# $task0 = mask 0x1 (bit 0)

main:
SCANMED $report0      # Load medbay status into $report0
TASKAND $report0, $report0, $task0 # Keep only bit 0 (confirmed bit)
SUSBEQ $report0, $void, maybe_sus # If bit 0 == 0, branch (not confirmed)

# Confirmed crew → do nothing
VENT done

maybe_sus:
VOTEOUT $crewA        # Mark this player as ejected in the player table
CLEARVOTES            # Reset any vote counters (cleanup)
done:
VENT done             # End / idle
```

### **What this shows:**

- SCANMED: reads a special I/O location for medbay status.
- TASKAND: uses a mask to isolate one bit.
- SUSBEQ with \$void (0) checks if that bit is 0.
- VOTEOUT: writes to PBASE + 4 \* playerId.
- CLEARVOTES: resets some predefined registers (like vote tallies).

### **Program 3 – Random Task + Panic Vent if Suspicion is Negative**

Goal: Assign a random task, mark it as fake, and if the suspicion meter is negative, run an emergency meeting.

Code:

```
# Assume:
# $task0 = suspicion meter (negative means danger)
# meeting_handler: label where meeting logic lives

main:
RANDOMTASK $task1      # Get a random task ID into $task1
FAKETASK $task2, $task1 # Copy task → $task2, set SF = 1 (fake task = sus)
```

```
VENTIFNEG $task0, call_meeting # If suspicion < 0, branch to call_meeting
VENT     done                # Otherwise, continue normal execution / end
```

call\_meeting:

```
EMERGENCY meeting_handler # Save return in $return, jump to meeting code
```

meeting\_handler:

# ... handle votes, checks, etc. (not expanded here)

```
VENT     meeting_handler # Simple loop placeholder
```

done:

```
VENT     done            # End / idle
```

### **What this shows:**

- RANDOMTASK: generates a pseudo-random task ID.
- FAKETASK: copies a value and sets SF = 1 (fake task → sus).
- VENTIFNEG: conditional branch based on a negative value (danger level).
- EMERGENCY: like a function call to an emergency meeting handler.