CENG 3420
Computer Organization & Design

Lecture 02: ISA Introduction

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Textbook: Chapters 1.3 & 2.1)
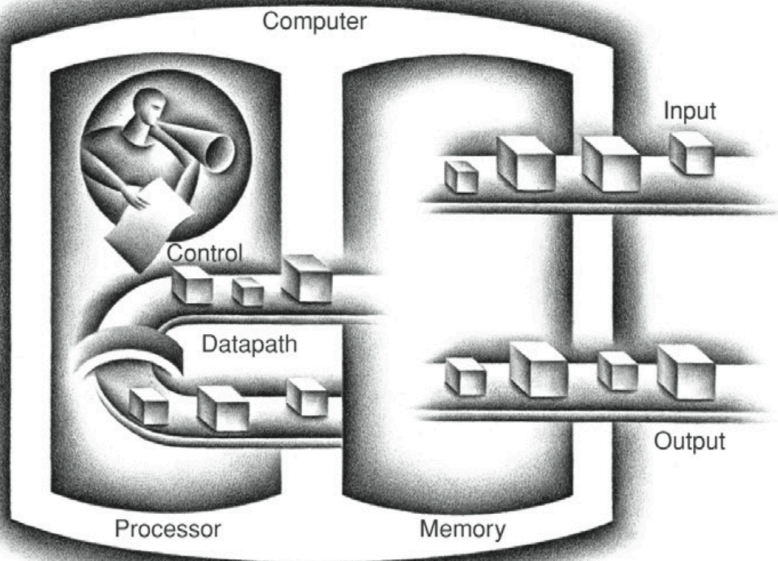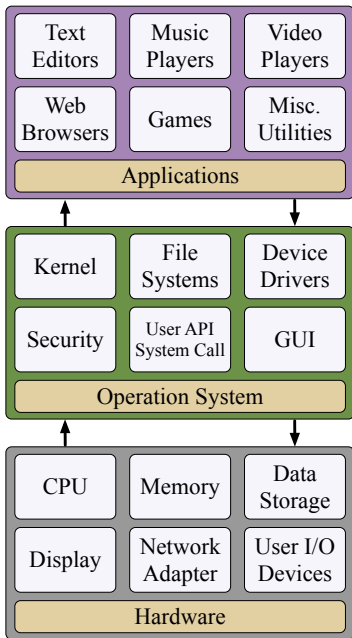
2024 Spring

Organization – First Glance

Components

- processor (datapath, control)
- input (mouse, keyboard)
- output (display, printer)
- memory (cache, main memory, disk drive, CD/DVD)
- network

Our primary focus: the processor (datapath and control) and its interaction with memory systems

- Implemented using tens/hundreds of millions of transistors
- Impossible to understand by looking at each transistor
- We need abstraction!

Text Editors	Music Players	Video Players
Web Browsers	Games	Misc. Utilities
Applications		

Kernel	File Systems	Device Drivers
Security	User API System Call	GUI
Operation System		

CPU	Memory	Data Storage
Display	Network Adapter	User I/O Devices
Hardware		

- Capabilities and performance characteristics of the principal Functional Units (FUs). (e.g., register file, ALU, multiplexors, memories, ...)

- The ways those FUs are interconnected (e.g., buses)

- Logic and means by which information flow between FUs is controlled

- The machine's Instruction Set Architecture (ISA)

- Register Transfer Level (RTL) machine description
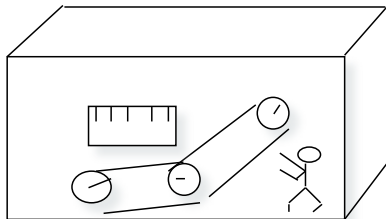
Control needs to have circuitry to

- Decide which is the next instruction and input it from memory
- Decode the instruction
- Issue signals that control the way information flows between datapath components
- Control what operations the datapath's functional units perform

Datapath needs to have circuitry to

- Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
- Interconnect the functional units so that the instructions can be executed as required
- Load data from and store data to memory

Applications software

Systems software
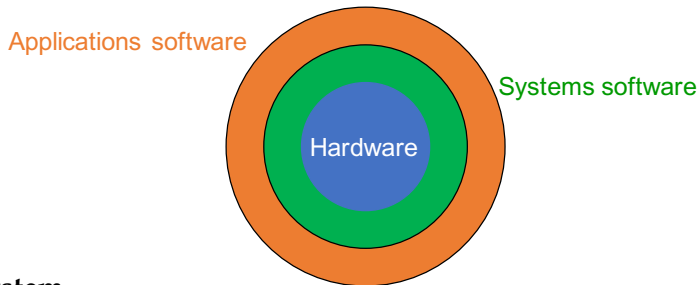
Hardware

Operating System

- Supervising program that interfaces the user's program with the hardware (e.g., Linux, iOS, Windows)

- Handles basic input and output operations

- Allocates storage and memory

- Provides for protected sharing among multiple applications

Compiler

- Translate programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute
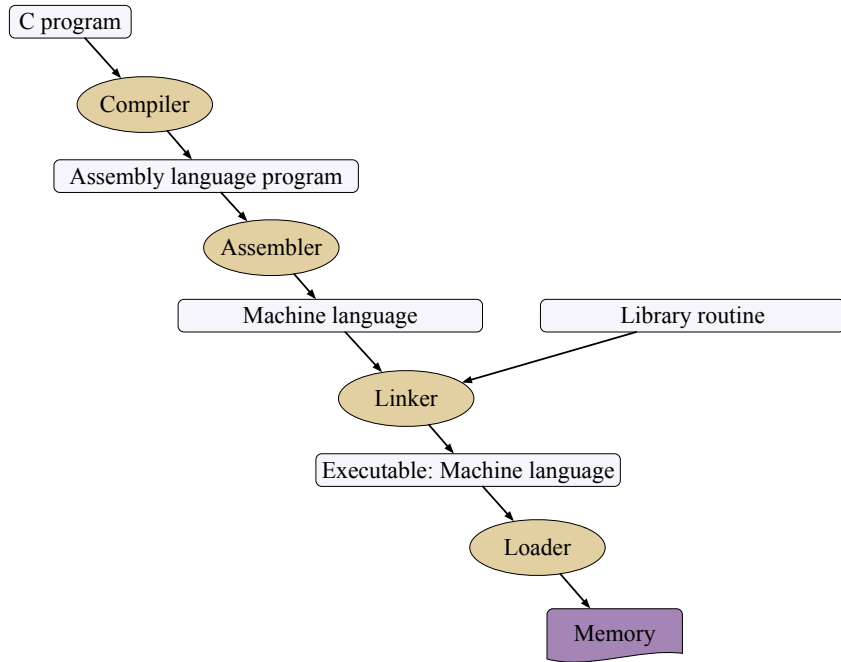
Advantages of Higher-Level Languages ?

- Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, ...)

- Improve programmer productivity – more understandable code that is easier to debug and validate

- Improve program maintainability

- Allow programs to be independent of the computer on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)

- Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine

As a result, very little programming is done today at the assembler level

```
C program
   │
   ▼
Compiler
   │
   ▼
Assembly language program
   │
   ▼
Assembler
   │
   ▼
Machine language        Library routine
        \              /
         ▼            ▼
            Linker
              │
              ▼
Executable: Machine language
              │
              ▼
           Loader
              │
              ▼
           Memory
```

- High-level language program (in C)

```
swap (int v[], int k)
(int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
)
```

one-to-many

C compiler

- Assembly language program

```
swap: sll   $2, $5, 2
      add   $2, $4, $2
      lw    $15, 0($2)
      lw    $16, 4($2)
      sw    $16, 0($2)
      sw    $15, 4($2)
      jr    $31
```

one-to-one

assembler

- Machine (object) code

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
 . . .
```
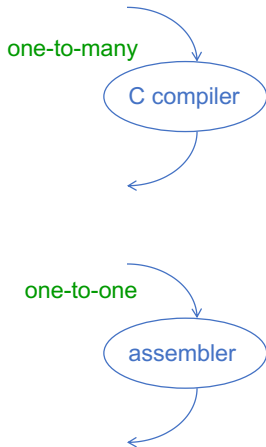
- High-level language program (in C)

```
swap (int v[], int k)
(int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
)
```

one-to-many

C compiler

- Assembly language program

```
swap:   sll     $2, $5, 2
        add     $2, $4, $2
        lw      $15, 0($2)
        lw      $16, 4($2)
        sw      $16, 0($2)
        sw      $15, 4($2)
        jr      $31
```

one-to-one
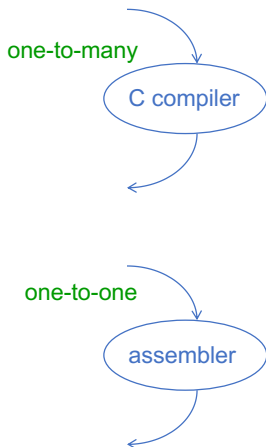
assembler

- Machine (object) code

```
000000 00000 00101 0001000010000000
000000 00100 00010 00010 00000100000
. . .
```
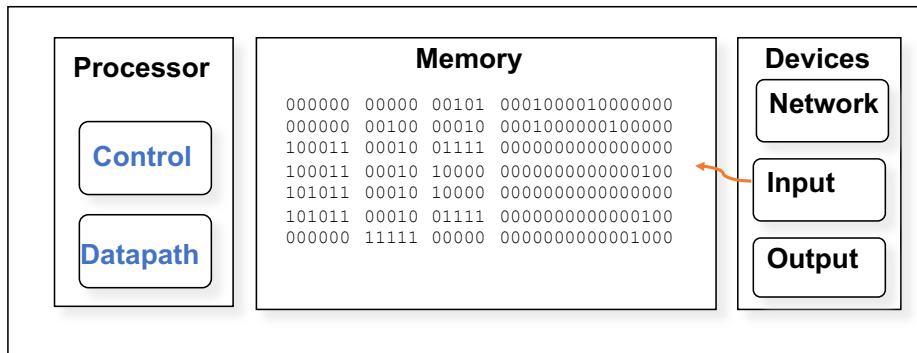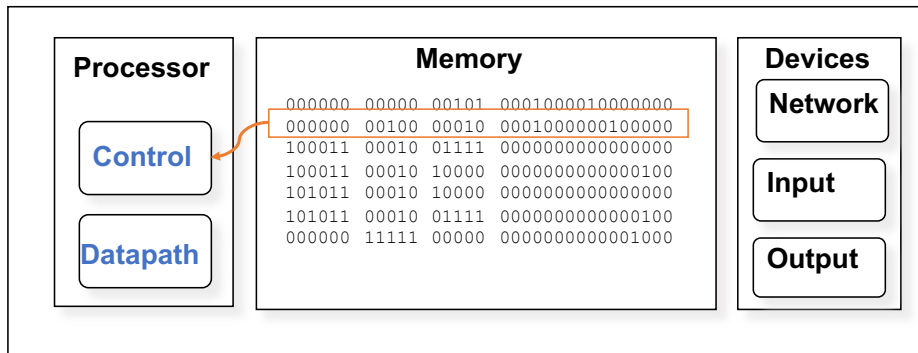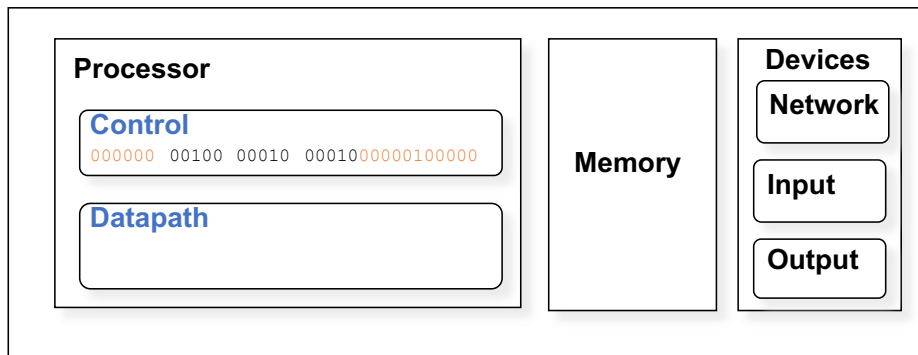
Max # of operations?

```
000000  00000  00101  0001000010000000
000000  00100  00010  0001000000100000
100011  00010  01111  0000000000000000
100011  00010  10000  0000000000000100
101011  00010  10000  0000000000000000
101011  00010  01111  0000000000000100
000000  11111  00000  0000000000001000
```

Processor

Control

Datapath

Memory

Devices

Network

Input

Output

Processor

Control

Datapath

Memory

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```
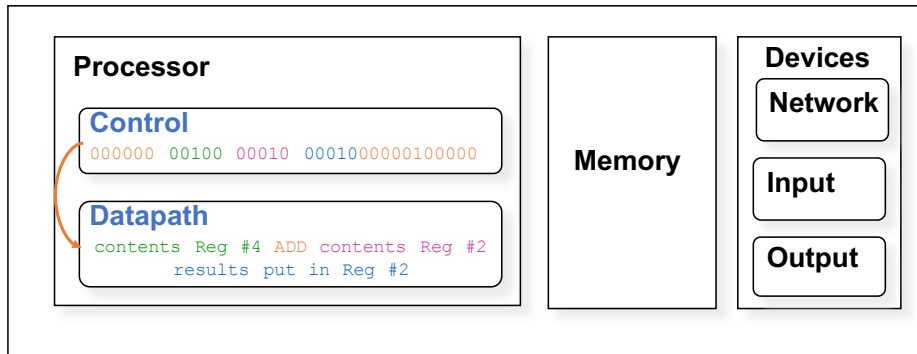
Devices

Network

Input

Output
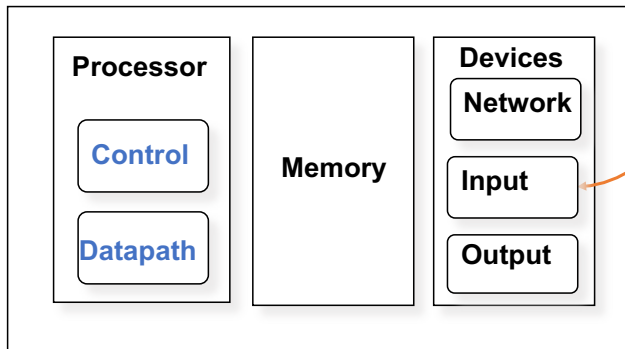
Processor fetches an instruction from memory

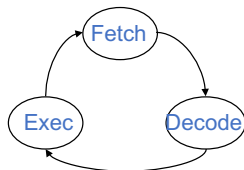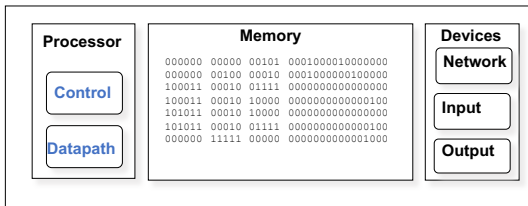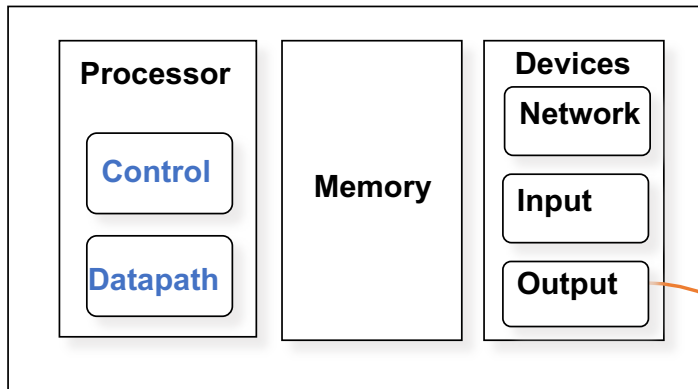- Control decodes the instruction to determine what to execute

- Control decodes the instruction to determine what to execute
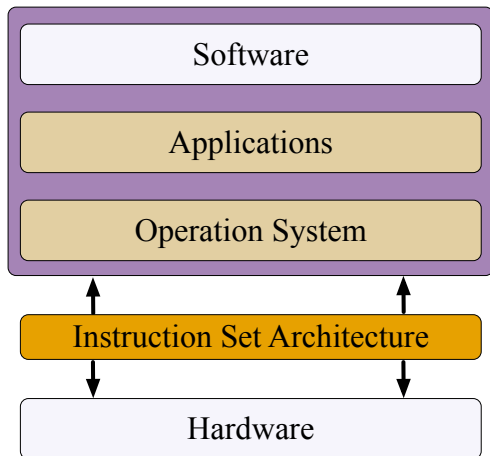- Datapath executes the instruction as directed by control

- Processor fetches the next instruction from memory
- How does it know which location in memory to fetch from next?
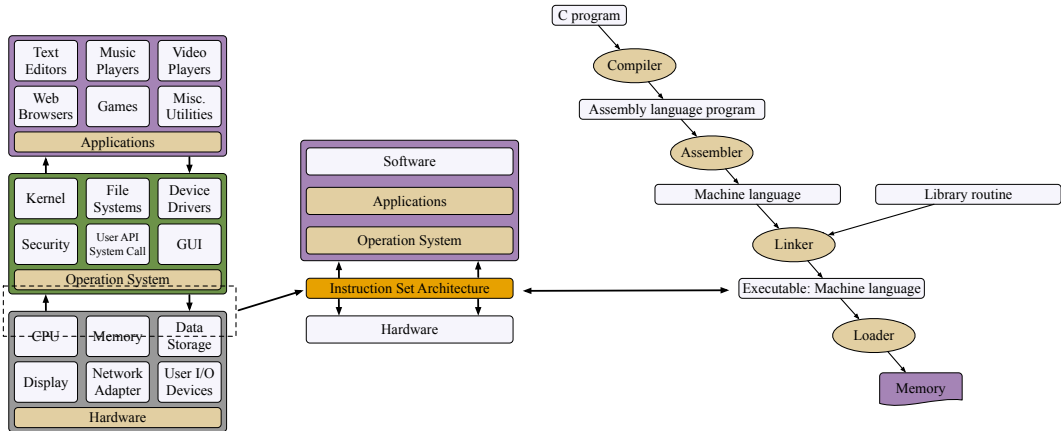
```
0000010001010000000000000000000000
0000000001001111000000000000000100
0000001111100000000000000000001000
```
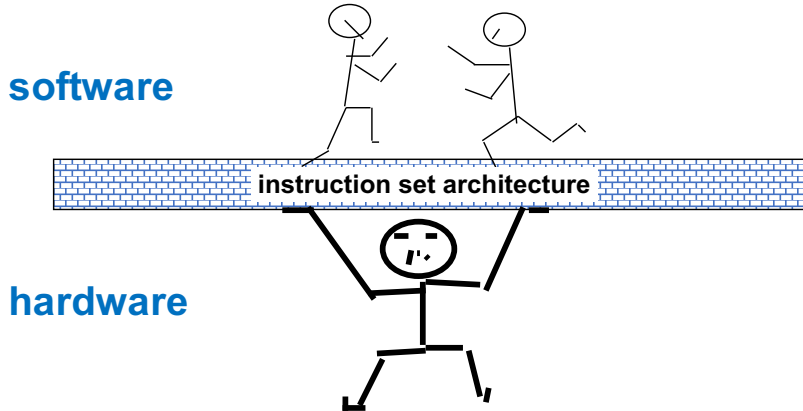
ISA

Instruction Set Architecture (ISA)

The interface description separating the software and hardware

software

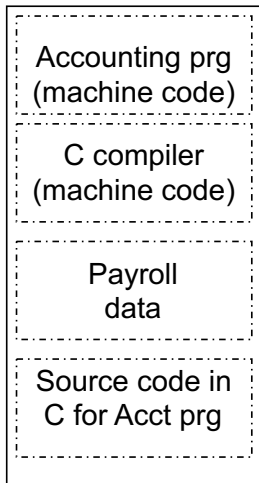**instruction set architecture**

hardware

- ISA, or simply architecture – the abstract interface between the hardware and the lowest level software that includes all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, ...

- Enables implementations of varying cost and performance to run identical software

- The combination of the basic instruction set (the ISA) and the operating system interface is called the application binary interface (ABI)

- ABI: The user portion of the instruction set plus the operating system interfaces used by application programmers. Defines a standard for binary portability across computers.

① Instructions are represented as numbers and, as such, are indistinguishable from data

② Programs are stored in alterable memory (that can be read or written to) just like data

Memory

Accounting prg (machine code)
C compiler (machine code)
Payroll data
Source code in C for Acct prg

Stored-Program Concept

- Programs can be shipped as files of binary numbers – binary compatibility

- Computers can inherit ready-made software provided they are compatible with an existing ISA – leads industry to align around a small number of ISAs

The language of the machine

- Want an ISA that makes it easy to build the hardware and the compiler while maximizing performance and minimizing cost

Our target: the RISC-V ISA

- similar to other ISAs developed since the 1980's

- RISC-V is originated from MIPS, the latter of which is used by Broadcom, Cisco, NEC, Nintendo, Sony, ...

Design Goals

Maximize performance, minimize cost, reduce design time (time-to-market), minimize memory space (embedded systems), minimize power consumption (mobile systems)

RISC-V

Complex Instruction Set Computer (CISC)

Lots of instructions of variable size, very memory optimal, typically less registers.

- Intel x86

Reduced Instruction Set Computer (RISC)

Instructions, all of a fixed size, more registers, optimized for speed. Usually called a "Load/Store" architecture.

- RISC-V, LC-3b, MIPS, ARM, Sun SPARC, HP PA-RISC, IBM PowerPC ...

- Used in many embedded systems
- E.g., Nintendo-64, Playstation 1, Playstation 2

RISC Philosophy

- fixed instruction lengths
- load-store instruction sets
- limited number of addressing modes
- limited number of operations

- Instruction sets are measured by how well compilers use them as opposed to how well assembly language programmers use them

Simplicity favors regularity

- fixed size instructions
- small number of instruction formats
- opcode always the first 6 bits

Smaller is faster

- limited instruction set
- limited number of registers in register file
- limited number of addressing modes

Make the common case fast

- arithmetic operands from the register file (load-store machine)
- allow instructions to contain immediate operands

Good design demands good compromises

- For RV32I, 4 base instruction formats (R/I/S/U) and 2 extended instruction formats (B/J)

RISC-V

- An open standard instruction set architecture (ISA)
- A clean break from the earlier MIPS-inspired designs
- Modular ISA organization
- Open standards, numerous proprietary and open-source cores
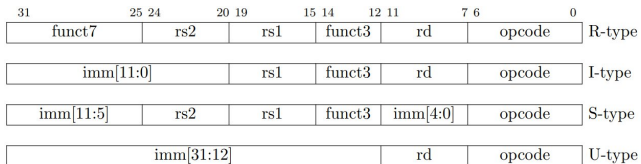- Managed by RISC-V Foundation

Instruction Categories

- Load and Store instructions
- Bitwise instructions
- Arithmetic instructions
- Control transfer instructions
- Pseudo instructions

4 Base Instruction Formats: all 32 bits wide

31	25	24	20	19	15	14	12	11	7	6	0	
funct7		rs2		rs1		funct3		rd		opcode		R-type

imm[11:0]			rs1		funct3		rd		opcode		I-type

imm[11:5]		rs2		rs1		funct3		imm[4:0]		opcode		S-type

imm[31:12]				rd		opcode		U-type

Register Names	ABI Names	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary / Alternate link register
x6-7	t1 - t2	Temporary register
x8	s0 / fp	Saved register / Frame pointer
x9	s1	Saved register
x10-11	a0-a1	Function argument / Return value registers
x12-17	a2-a7	Function argument registers
x18-27	s2-s11	Saved registers
x28-31	t3-t6	Temporary registers