SAP 1, designed by Albert Paul Malvino in *Digital Computer Electronics*, is a very simple and intuitive computer architecture. The goal of this project is to implement SAP 1 using logic gates with as less abstraction as possible. The software to open SAP1 is [Logisim](http://www.cburch.com/logisim/).

**SAP 1 Architecture**

There are 8 main components in SAP 1 architecture:

* **PC** (Program Counter): 4 bit, counting up when Count signal is enabled;
* **MAR** (Memory Address Register): 4 bit, loading addresses from PC through the signal bus;
* **PROM** (Programmable ROM): 16 bytes, data can be entered manually when in Program Mode
* **IR** (Instruction Register): 4-bit, receiving opcodes from the signal bus, the opcodes then go to the Instruction Decoder and the instruction goes into the Control Unit;
* **AC** (Accumulator): 8-bit register, connected to the three-state data bus and the two-state ALU;
* **ALU**: 8-bit, two inputs are from AC and B Register;
* **BReg**: 8-bit, reading data from data bus and output to ALU;
* **CON** (Control Unit): generating a 12-bit control word (HLT not included), depends on the instruction. The 13-bit control word determines how the registers and different components cooperate and react to the next positive clock edge.

**Control Word**

The 12-bit control word coming out of the Control Unit is CON = Cp Ep Lm Er Li Ei La Ea Su Eu Lb Lo:

**Cp**: PC count enable, the PC increments by one when Cp is high (1);

**Ep**: Enable data in PC goes out to the bus;

**Lm**: Load the data on the bus into MAR;

**Er**: Enable data in PROM output to the bus;

**Li**: Load data from the bus to IR;

**Ei**: Enable data in IR output to the Instruction Decoder;

**La**: Load data on the bus into AC;

**Ea**: Enable data in AC output to the three-state data bus and two-state ALU;

**Su**: The subtract signal;

**Eu**: The add signal;

**Lb**: Load B Register;

**Lo**: Output the data from AC;

**HLT**: Stop the clock signal.

**Fetch-Decode-Execute Cycle**

The Instruction Decoder generates five instructions: Load (0000), Add (0001), Subt (0010), Output (1110), and HLT (1111).

There are 6 phases for every fetch-decode-execute cycle, which is achieved by a 6-bit ring counter (T0 – T5):

Fetch Cycle:

Address Phase (**T0**): Ep and Lm are high, MAR 🡨 PC;

Memory Phase (**T1**): Er and Li are high, IR 🡨 M[MAR];

Increment Phase (**T2**): Cp is high, PC 🡨 PC + 1;

Execution Cycle (Decode included):

e.g. **Load**:

**T3**: Ei and Lm go high, the opcode is sent to Instruction Decoder and the address field is loaded into MAR, MAR🡨 IR[3-0];

**T4**: Er and La go high, AC 🡨 M[MAR];

**T5**: Nop, do-nothing phase.

e.g. **Add**:

**T3**: Ei and Lm go high, the opcode is decoded and MAR🡨 IR[3-0];

**T4**: Er and Lb go high, B REG 🡨 M[MAR];

**T5**: Eu and La go high, AC 🡨 AC + M[MAR].

The microinstructions of fetch cycle—T0 – T2—are identical for all the instructions. The microinstruction through T3 – T5 depends on the decoded instruction (see the table below):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Opcodes**  **(Binary)** | **Assembly** | **T0** | **T1** | **T3** | **T3** | **T4** | **T5** |
| 0000 | Load | Ep, Lm | Er, Li | Cp | Ei, Lm | Er, La | nop |
| 0001 | Add | Ep, Lm | Er, Li | Cp | Ei, Lm | Er, Lb | Eu, La |
| 0010 | Subt | Ep, Lm | Er, Li | Cp | Ei, Lm | Er, Lb | Su, La |
| 1110 | Output | Ep, Lm | Er, Li | Cp | Ea, Lo | Eo, Lport | nop |
| 1111 | Halt | Ep, Lm | Er, Li | Cp | nop | nop | halt |

**The Terminal and Sample Programs**

There are 4 input pins to program, delete, start, control, and stop a stored program.

Press and release ***Reset*** to start/restart a program. Changing ***Trace*** to high (1) will pause the clock so steping through a program is possible by controlling Trace button; Changing ***Trace*** to low (0) will go back to the normal flow of the program. ***Delete Prog*** will clear everything stored in PROM. To enter a program, the ***Program Mode*** needs to be high (1), which is indicated by the Prog Mode LED. The other indication LEDs or output LEDs are self-explanatory.

The following sample program adds 240 in decimal to 186 (which cause overflow) then subtracts 255, which equals to 171.

1. Click ***Program Mode*** so that it changes to 1 and the ***Program Mode*** LED is on;
2. Enter 0000 1000 (Load 1000) at **Address** 0000 (the right most is the 1st bit and the left most is the 7th bit), press ***Write*** button;
3. Release the ***Write*** button, change ***Address*** to 0001, enter 0001 1001 (Add 1001), then press ***Write***;
4. Release the ***Write*** button, change ***Address*** to 0010, enter 0010 1010 (Subt 1010), then press ***Write***;
5. Release the ***Write*** button, change ***Address*** to 0011, enter 1110 0000(Output, the right most four digits don’t matter), press ***Write***;
6. Release the ***Write*** button, change ***Address*** to 0100, enter 1111 0000 (HLT, the right most four digits don’t matter), press ***Write***;
7. Release the ***Write*** button, change ***Address*** to 1000, enter 1111 0000 (240), press ***Write***;
8. Release the ***Write*** button, change ***Address*** to 1001, enter 1011 1010 (186), press ***Write***;
9. Release the ***Write*** button, change ***Address*** to 1010, enter 1111 1111 (255), press ***Write***;
10. Release the ***Write*** button, press and release the ***Reset*** button to start the program.

|  |  |  |  |
| --- | --- | --- | --- |
| **240 + 186 = 426 – 255 = 171** | | | |
| Addr | Binary | Assembly | Comments |
| 0000 | 0000 1000 | Load 1000 |  |
| 0001 | 0001 1001 | Add 1001 |  |
| 0010 | 0010 1010 | Subt 1010 |  |
| 0011 | 1110 xxxx | Out | Output (10101011) |
| 0100 | 1111 xxxx | HLT |  |
| 0101 |  |  |  |
| 0110 |  |  |  |
| 0111 |  |  |  |
| 1000 | 11110000 |  | 240 |
| 1001 | 10111010 |  | 186 |
| 1010 | 11111111 |  | 255 |

Some other sample programs see the ***Sample Programs*** word file.