**Assignment 4 “Procedure and Function Calls”**

**Group - SS05**

**The CPU is based on MIPS architecture.**

* **Memory:** 2048 locations of 1 bytes each (mem[2048]) = 2KB of memory

|  |  |  |  |
| --- | --- | --- | --- |
| **Memory** | **Memory Address** | **Address in program** | **Locations (bytes)** |
| Data memory | 0x0481 to 0x07FF | mem[1053] to mem[2047] | 896 |
| Stack | 0x0400 to 0x0480 | mem[1024] to mem[1152] | 128 |
| Instruction memory | 0x0200 to 0x03FF | mem[512] to mem[1023] | 512 |
| OS memory | 0x0000 to 0x01FF | mem[0] to mem[511] | 512 |

* **Instruction size:** 4 bytes = 32 bits

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode (8 bits) | Operand 1 (8 bits) | Operand 2 (8 bits) | Operand 3 (8 bits) |

* **Opcodes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Opcode** | **Instruction** | **Opcode** |
| r0 | 0x00 | lw | 0x00 |
| r1 | 0x01 | sw | 0x01 |
| r2 | 0x02 | add | 0x02 |
| r3 | 0x03 | sub | 0x03 |
| r4 | 0x04 | mul | 0x04 |
| r5 | 0x05 | div | 0x05 |
| r6 | 0x06 | mod | 0x06 |
| r7 | 0x07 | push | 0x07 |
| r8 | 0x08 | pop | 0x08 |
| r9 | 0x09 | lea | 0x09 |
| r10 | 0x0A | beq | 0x0A |
| r11 | 0x0B | bne | 0x0B |
| r12 | 0x0C | slt | 0x0C |
| r13 | 0x0D | j | 0x0D |
| r14 | 0x0E | jr | 0x0E |
| r15 | 0x0F | mov | 0x0F |
|  |  | mvi | 0x10 |
|  |  | inc | 0x11 |
|  |  | dec | 0x12 |
|  |  | lwr | 0x13 |
|  |  | ex | 0x14 |
|  |  | jal | 0x15 |
|  |  | ret | 0x16 |

* **Status register:** 8bits

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| - | - | OF | SF | ZF | AC | PF | CF |
| (Bit 7) | (Bit 6) | (Bit 5) | (Bit 4) | (Bit 3) | (Bit 2) | (Bit 1) | (Bit 0) |

**NOTE:** As procedure call’s parameters gets stored in registers r10, r11, r12 at time of user’s input, instructions setting these parameters are not considered in assembly language program and it directly starts by calling binary search function.

Parameters for binary search function:

r10 = Key value to be searched

r11 = Starting index of array (0)

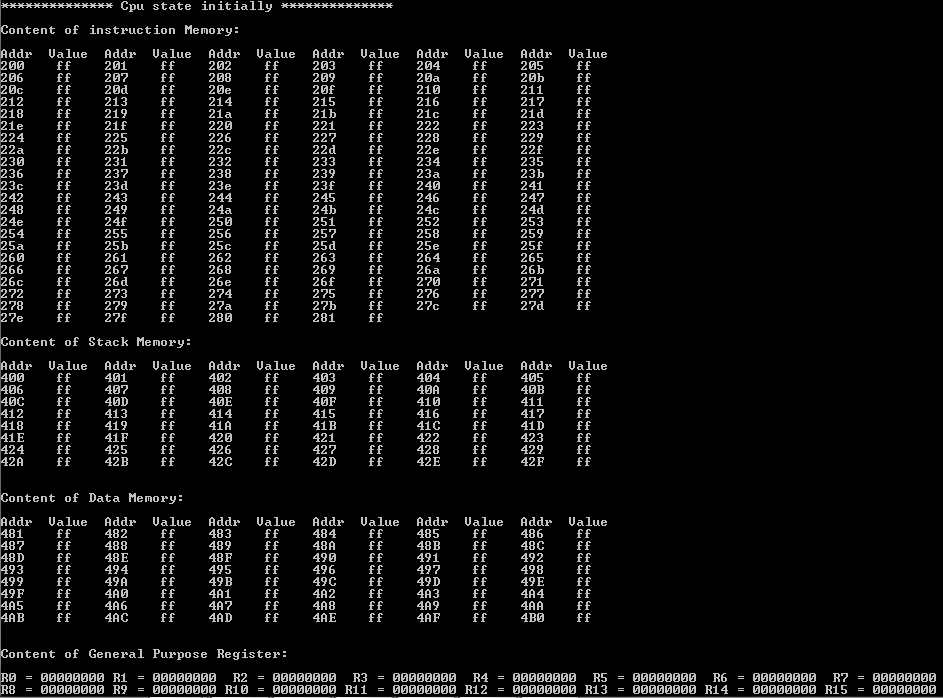
r12 = Ending index of array (N) = Number of elements

r14 = Result of binary search function. Returns index of the element if key value found, returns -1 (0xFFFFFFFF) if key value not found in the array.

**Assembly program description**

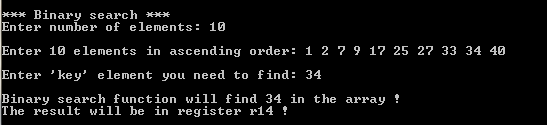
|  |  |  |
| --- | --- | --- |
| **Assembly program** | | **Operation** |
|  | jal lb0 | Call binary search function (function parameters are stored in r10,r11,r12) |
|  | ex 1 | Exit program |
| lb0: | slt r1,r12,r11 | Set r1 if Starting index > Ending index |
|  | mvi r2,0001 | Store 1 in register r2 |
|  | beq r1,r2,lb1 | Branch to lb1 if r1=r2 |
|  | add r3,r11,r12 | r3 = Starting index + Ending index |
|  | mvi r2,0002 | Store 2 in r2 |
|  | div r3,r3,r2 | r3 = (Starting index + Ending index) / 2 |
|  | mvi r5,0481 | Store base address 0x0481 in r5 |
|  | mvi r6,0004 | Store word length 4 in r6 |
|  | lea r5,r3,r6 | Load effect address in r5. (0x0481 + mid \* 4) |
|  | lwr r7,r5 | Store value pointed by r5 |
|  | bne r7,r10,lb2 | Check (Mid value == key value) else jump to lb2 |
|  | mov r14,r3 | Store index value in r14 |
|  | ret 1 | Return from function (pops return address from stack and copy to PC) |
| lb2: | slt r1,r10,r7 | Set r1 if key < mid index |
|  | mvi r2,0001 | Store 1 in register r2 |
|  | beq r1,r2,lb3 | Check (r1==r2) If equal go to lb3 |
|  | add r11,r3,r2 | Add 1 in Starting index (Starting index = Mid index + 1) |
|  | jal lb0 | Recursive call to binary search function |
|  | ret 1 | Return from function (pops return address from stack and copy to PC) |
| lb3: | sub r12,r3,r2 | Subtract 1 from Ending index (Ending index = Mid index - 1) |
|  | jal lb0 | Recursive call to binary search function |
|  | ret 1 | Return from function (pops return address from stack and copy to PC) |
| lb1: | sub r14,r14,r2 | Subtract r2 from r14 means -1 in r14 |
|  | ret 1 | Return from function (pops return address from stack and copy to PC) |

* **CPU State initially**

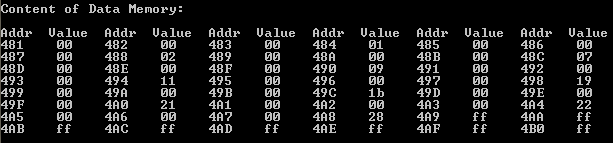


**CASE 1:**

* **User input (number of elements even, key > mid value)**

****

* **User array values in data memory**

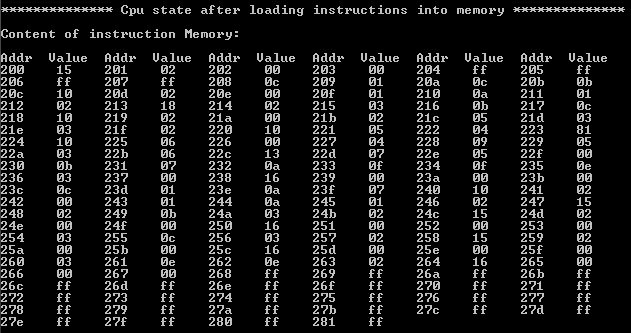
****

* **Binary search function parameters in r10, r11 and r12**

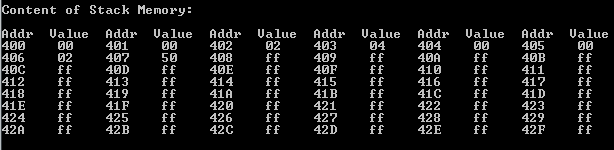
****

* **CPU State after loading instructions in memory (Big Endian Machine)**

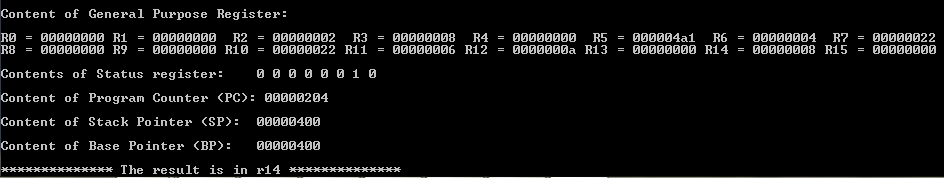
****

****

* **Stack memory holds return addresses as function calls recursively**

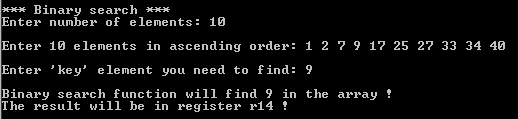
****

* **Result in r14 = 0x00000008 (number 34 found at 8th index in an array)**

****

**CASE 2:**

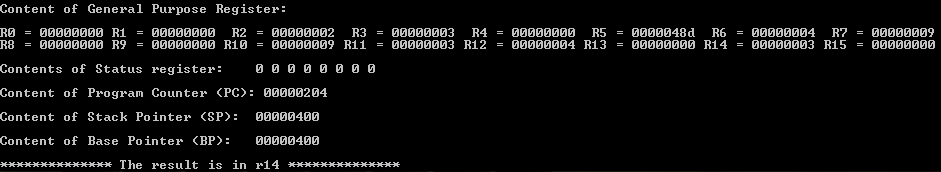
* **User input (number of elements even, key < mid value)**

****

* **Binary search function parameters in r10, r11 and r12**

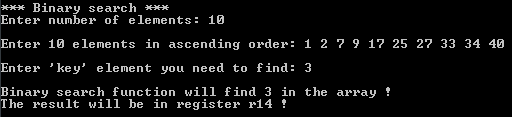
****

* **Result in r14 = 0x00000003 (number 9 found at 3rd index in an array)**

****

**CASE 3:**

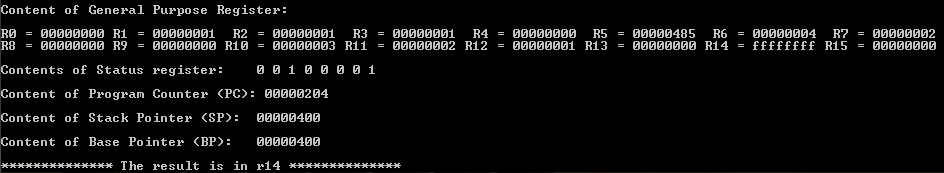
* **User input (number of elements even, key not in an array)**

****

* **Binary search function parameters in r10, r11 and r12**

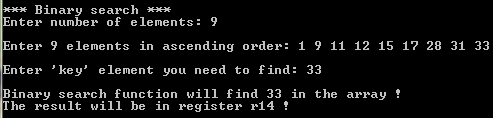
****

* **Result in r14 = 0xFFFFFFFF = -1 (number 3 not found in an array)**

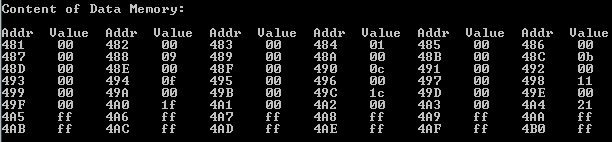
****

**CASE 4:**

* **User input (number of elements odd, key > mid value)**

****

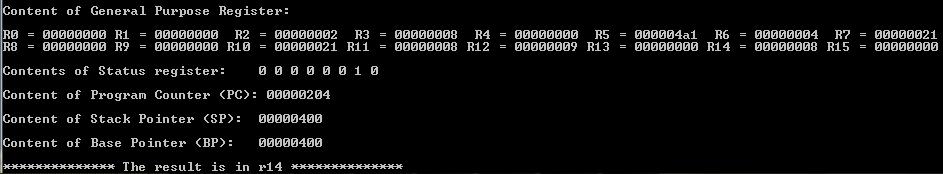
* **User array values in data memory**

****

* **Binary search function parameters in r10, r11 and r12**

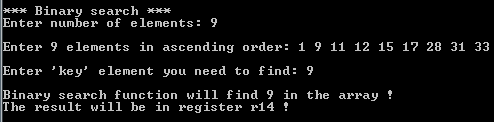
****

* **Result in r14 = 0x00000008 (number 33 found at 8th index in an array)**

****

**CASE 5:**

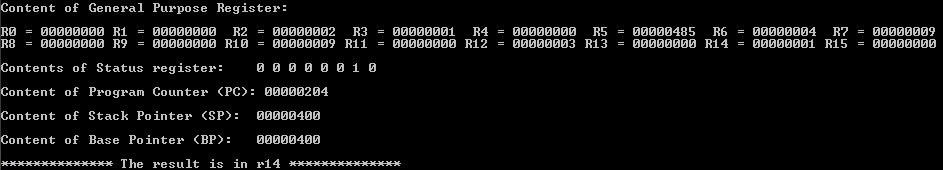
* **User input (number of elements odd, key < mid value)**

****

* **Binary search function parameters in r10, r11 and r12**

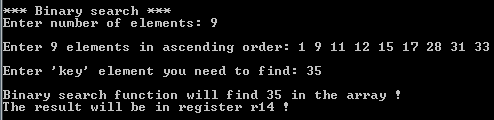
****

* **Result in r14 = 0x00000001 (number 9 found at 1st index in an array)**

****

**CASE 6:**

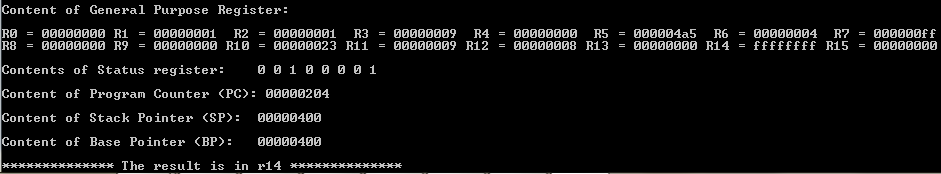
* **User input (number of elements odd, key not in an array)**

****

* **Binary search function parameters in r10, r11 and r12**

****

* **Result in r14 = 0xFFFFFFFF = -1 (number 35 not found in an array)**

****