I implement the AND, OR, XOR, ANDI, ORI, XORI, ADD, SUB, ADDI, LW, SW, BEQ, BNE, BLT, BGE, SLL, SRL, SRA, SLLI, SRLI, SRAI, JAL, JALR, LH, LB, SH and SB assembly instructions for lab 4.1.

In total, I implement \_\_27\_\_ assembly instructions. So, I would write C programs that use at least \_\_19\_\_ different assembly instructions in total.

## Turn k bit off (ADDI / SLL / XORI / AND / JALR)

|  |  |  |
| --- | --- | --- |
| int turn\_k\_bit\_off(  const int n, const short k  ) {  return n & ~(1 << (k - 1));  } | int turn\_k\_bit\_off(  const int n, const short k  ) {  int a1 = k-1;  a1 = 1 << a1;  a1 = ~a1;  a0 = n & a1  return a0;  } | turn\_k\_bit\_off:  ADDI a1, a1, -1  ADDI a2, x0, 1  SLL a1, a2, a1  XORI a1, a1, -1  AND a0, a0, a1  JALR x0, x1, 0 |

## String Concatenation (BEQ / BNE / LBU / SB)

Assume variable “a” is large enough to contain a + b + ‘\0’.

|  |  |  |
| --- | --- | --- |
| char \*strcat(char \*a, const char \*b) {  char \*ret = a;  while (\*a != '\0')  ++a;  while (\*b != '\0')  \*a++ = \*b++;  \*a = '\0';  return ret;  } | char \*strcat(char \*a, const char \*b) {  char \*a2 = a;  if (\*a == ‘\0’) goto end1;  start1:  a += 1;  if (\*a != ‘\0’) goto start1;  end1:  if (\*b == ‘\0’) goto end2;  start2:  \*a = \*b;  a++;  b++;  if (\*b != ‘\0’) goto start2;  end2:  \*a = ‘\0’;  return a2;  } | strcat:  ADDI a2, a0, 0  LBU a3, 0(a0)  BEQ a3, x0, end1  start1:  ADDI a0, a0, 1  LBU a3, 0(a0)  BNE a3, x0, start1  end1:  LBU a4, 0(a1)  BEQ a4, x0, end1  start2:  LBU a4, 0(a1)  SB a4, 0(a0)  ADDI a0, a0, 1  ADDI a1, a1, 1  LBU a4, 0(a1)  BNE a4, x0, start2  end2:  SB x0, 0(a0)  ADDI a2, a0, 0  JALR x0, x1, 0 |

Memory Allocation

Stack Registers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | \*b | a4 |
|  |  |  |  | \*a | a3 |
|  |  |  |  | a2 | a2 |
|  |  |  |  | b | a1 |
|  |  |  |  | a | a0 |

…

Template:

## \_\_code\_purpose\_\_ (new assembly instructions)

|  |  |  |
| --- | --- | --- |
| C code | Simplify C | Assembly |

Optional part for each code above:

Memory Allocation

Stack Registers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | \*b | a4 |
|  |  |  |  | \*a | a3 |
|  |  |  |  | a2 | a2 |
|  |  |  |  | b | a1 |
|  |  |  |  | a | a0 |