**Inline Assembly**

|  |
| --- |
| **\_\_asm** int sum4(int a, int b, int c, int d){  // arguments stored in r0-r3  **PUSH {r4, lr}**  MOV r4, r0 ; r0 = 1st argument  ADD r4, r4, r1 ; r1 = 2nd argument  ADD r4, r4, r2 ; r2 = 3rd argument  ADD r0, r4, r3 ; r3 = 4th argument, r0 = return  **POP {r4, pc}**  }  int main(void){  int s = sum4(1, 2, 3, 4);  while(1);  } |

|  |
| --- |
| int sum4(int a, int b, int c, int d){  int t;  **\_\_asm** {  ADD t, a, b; // t, a, and b are virtual registers  ADD t, c; // Cannot directly access r0-r15  ADD t, d; // Have to use c comment style  }  return t;  }  int main(void){  int s = sum4(1, 2, 3, 4);  while(1);  } |

**C program calls assembly subroutines**

|  |  |
| --- | --- |
| **C Program (main.c)** | **Assembly Program (strcpy.s)** |
| char src[25] = "Hello!";  char dst[25];  **extern void strcpy(char\* dst, char\* src);**  int main(void){  strcpy(dst, src);  while(1);  } | AREA stringCopy, CODE  **EXPORT strcpy**  ALIGN  strcpy PROC  loop LDRB r2, [r1] ; Load a byte, r0 = dst  STRB r2, [r0] ; Store a byte, r1 = src  ADD r1, #1 ; Increase memory pointer  ADD r0, #1 ; Increase memory pointer  CMP r2, #0 ; Zero terminator  BNE loop ; Loop if not null terminator  **BX lr ; Return**  ENDP  END |

|  |  |
| --- | --- |
| **C Program (Main.c)** | **Assembly Program (count.s)** |
| int counter;  **extern int getValue();**  **extern void increment();**  **extern void setValue(int c);**  int main(void) {  int c = 0;  setValue(1);  increment();  c = getValue();  c = c + 1;  while(1);  } | AREA count, CODE  **IMPORT counter**  ALIGN  ENTRY  setValue PROC  **EXPORT setValue**  LDR r1, =counter  STR r0, [r1]  BX lr  ENDP  getValue PROC  **EXPORT getValue**  LDR r1, =counter  LDR r0, [r1]  BX lr  ENDP    increment PROC  **EXPORT increment**  LDR r1, =counter  LDR r0, [r1]  ADD r0, r0, #1  STR r0, [r1]  BX lr  ENDP    END |

**Assembly program calls C functions.**

|  |  |
| --- | --- |
| **Assembly Program** | **C Program** |
| AREA main, CODE  EXPORT \_\_main  ***IMPORT getValue***  ***IMPORT increment***  ***IMPORT setValue***  ALIGN  ENTRY  \_\_main  MOVS r2,#0x00  MOVS r0,#0x01  **BL setValue**  **BL increment**  **BL getValue**  MOV r2,r0    stop B stop  AREA myData, DATA  ***EXPORT counter***  counter DCD 0  END | **extern int counter;**  int getValue() {  return counter;  }  void increment() {  counter++;  }  void setValue(int c) {  counter = c;  } |

|  |  |
| --- | --- |
| C Program | Assembly Program |
| int isPal(int);  int main(){  int n;  n = isPal(12321);  while(1);  }  // Check palindrome number  int isPal(int n){  **static** int sum = 0;  int r;  if(n!=0) {  r = n % 10;  sum = sum\*10 + r;  isPal (n/10);  }    if (sum == n)  return 1;  else  return 0;  } | AREA palindrome, CODE  EXPORT \_\_main  ALIGN  ENTRY  \_\_main PROC  LDR r0,=12321  BL isPal  stop B stop  ENDP  ; Recursively check palindrome  isPal PROC  PUSH {r4,lr}  MOV r4,r0  CBZ r4,done ; if n is zero, done  MOV r2,#10  SDIV r1,r4,r2 ; r1 = n/10  MLS r3,r1,r2,r4 ; r3 = r = n - r1 \* 10;  LDR r1,=sum  LDR r1,[r1] ; r1 = sum  ADD r1,r1,r1,LSL #2 ; r1 = 5\*sum  ADD r1,r3,r1,LSL #1 ; sum = sum\*10 + r;  LDR r2,=sum  STR r1,[r2] ; save sum  MOV r2,#10  SDIV r0,r4,r2 ; r0 = n/10  BL isPal ; recursive call  done LDR r1,=sum  LDR r1,[r1]  CMP r1,r4  BNE no  yes MOV r0,#1 ; if palindrome  B exit  no MOV r0,#0 ; if not palindrome  exit POP {r4,pc}  ENDP    AREA myData, DATA  ALIGN  sum DCD 0  END |