HPY201

C → CLANG → MIPS Assembly

Vatsolakis Charalampos, 2018

Operations with Arrays in C

```
#include <stdio.h>

int A[4][4], B[4][4], C[4][4];

void readArrays(void){
   int i, j;
   for(i=0; i<4; i++){
      for(j=0; j<4; j++){
           scanf("%d", &(A[i][j]));
           scanf("%d", &(B[i][j]));
      }
   }
}



void printArray(void){
   int i, j;
   for(i=0; i<4; i++)
      for(j=0; j<4; j++)
         printf("%d ", C[i][j]);
}
```

```
void addArrays(void){
   int i, j;
   for(i=0; i<4; i++)
      for(j=0; j<4; j++)
          C[i][j]=A[i][j]+B[j][i];

}



int main(int args char* argv[]){
   readArrays();
   addArrays();
   printArray();

   return 0;

}
```

C → CLANG

```c
#include <stdio.h>


int A[4][4], B[4][4], C[4][4];


void readArrays(void){
   int i, j;
   for(i=0; i<4; i++){
      for(j=0; j<4; j++){
           scanf("%d", &(A[i][j]));
           scanf("%d", &(B[i][j]));
      }
   }
}
```

```c
#include <stdio.h>
long long R0=0, R1, R2, R3, R4, R5, R6,
    R7, R8, R9, R10, R11, R12;


int A[16], B[16], C[16];


void readArrays(void){
   R5 = 0;
   label_1:
   if(!(R5<16)) goto label_2;
      scanf("%d", &R2);
      A[R5] = (int)R2;
      scanf("%d", &R2);
      B[R5] = (int)R2;
      R5 = R5 + 1;
   goto label_1;
   label_2:
}
```

C → CLANG

```
void printArray(void){
   int i, j;
   for(i=0; i<4; i++)
      for(j=0; j<4; j++)
         printf("%d ", C[i][j]);
}
```

```
void printArray(void){
   R5 = 0;
   label_3:
   if(!(R5<16)) goto label_4;
      R6 = (long long)C[R5];
      printf("%d", (int)R6);
      printf(" ");
   R5 = R5 + 1;
   goto label_3;
   label_4:

}
```

C → intermediate C

```
void addArrays(void){
   int i, j;
   for(i=0; i<4; i++){
      for(j=0; j<4; j++){
          C[i][j]=
             A[i][j]
               +
             B[j][i];
       }
    }
}
```

```
void addArrays(void){
   int i;
   for(i=0; i<4; i++){
      int j;
      for(j=0; j<4; j++){
          C[(i<<2)+j] =
              A[(i<<2)+j]
                    +
              B[(j<<2)+i];
      }
   }
}
```

intermediate C → CLANG

```
void addArrays(void){
   int i;
   for(i=0; i<4; i++){
      int j;
      for(j=0; j<4; j++){
         C[(i<<2)+j] =
            A[(i<<2)+j]
                  +
            B[(j<<2)+i];
      }
   }
}
```

```
void addArrays(void){
   R5 = 0;
   label_5:
   if(!(R5<4)) goto label_6;
      R6 = 0;
      label_7:
      if(!(R6<4)) goto label_8;
         R7 = (R5<<2);
         R7 = R7 + R6;
         R7 = A[R7];
         R8 = (R6<<2);
         R8 = R8 + R5;
         R8 = B[R8];
         R7 = R7 + R8;
         R8 = (R5<<2);
         R8 = R8 + R6;
         C[R8] = R7;
      R6 = R6 + 1;
      goto label_7;
      label_8:
   R5 = R5 + 1;
   goto label_5;
   label_6:
}
```

intermediate C → CLANG

```
void addArrays(void){
   int i;
   for(i=0; i<4; i++){
      int j;
      for(j=0; j<4; j++){
         C[(i<<2)+j] =
            A[(i<<2)+j]
                  +
            B[(j<<2)+i];
      }
   }
}
```

```
void addArrays(void){
   R5 = 0;
   label_5:
   if(!(R5<4)) goto label_6;
      R6 = 0;
      label_7:
      if(!(R6<4)) goto label_8;
         R7 = (R5<<2);
         R7 = R7 + R6;
         R7 = A[R7];
         R8 = (R6<<2);
         R8 = R8 + R5;
         R8 = B[R8];
         R7 = R7 + R8;
         R8 = (R5<<2);
         R8 = R8 + R6;
         C[R8] = R7;
      R6 = R6 + 1;
      goto label_7;
      label_8:
   R5 = R5 + 1;
   goto label_5;
   label_6:
}
```

intermediate C → CLANG

```
void addArrays(void){
   int i;
   for(i=0; i<4; i++){
      int j;
      for(j=0; j<4; j++){
         C[(i<<2)+j] =
            A[(i<<2)+j]
                  +
            B[(j<<2)+i];
      }
   }
}
```

```
void addArrays(void){
   R5 = 0;
   label_5:
   if(!(R5<4)) goto label_6;
      R6 = 0;
      label_7:
      if(!(R6<4)) goto label_8;
         R7 = (R5<<2);
         R7 = R7 + R6;
         R7 = A[R7];
         R8 = (R6<<2);
         R8 = R8 + R5;
         R8 = B[R8];
         R7 = R7 + R8;
         R8 = (R5<<2);
         R8 = R8 + R6;
         C[R8] = R7;
      R6 = R6 + 1;
      goto label_7;
      label_8:
   R5 = R5 + 1;
   goto label_5;
   label_6:
}
```

CLANG → Assembly MIPS

```
#include <stdio.h>

long long R0=0, R1, R2, R3, R4, R5,
     R6, R7, R8, R9, R10, R11, R12;

int A[16], B[16], C[16];

void readArrays(void){
   R5 = 0;
   label_1:
   if(!(R5<16)) goto label_2;
      scanf("%d", &R2);
      A[R5] = (int)R2;
      scanf("%d", &R2);
      B[R5] = (int)R2;
      R5 = R5 + 1;
   goto label_1;
   label_2:
}
```

```
.data
.align 2
A: .space 64
.align 2
B: .space 64
.align 2
C: .space 64
.align 2
s: .asciiz " "
.text


readArrays:
add $5, $0, $0
label_1:
slti $11, $5, 16
beq $11, $0, label_2

addi $2, $0, 5
syscall

sll $6, $5, 2
la $7, A
add $6, $7, $6
sw $2, ($6)
```

CLANG → Assembly MIPS

```
void readArrays(void){
   R5 = 0;
   label_1:
   R11 = (R5<16);
   if(R11 == 0) goto label_2;
      scanf("%d", &R2);
      A[R5] = (int)R2;
      scanf("%d", &R2);
      B[R5] = (int)R2;
      R5 = R5 + 1;
   goto label_1;
   label_2:
}
```

```
readArrays:
add $5, $0, $0
label_1:
slti $11, $5, 16
beq $11, $0, label_2

addi $2, $0, 5
syscall

sll $6, $5, 2
la $7, A
add $6, $7, $6
sw $2, ($6)

addi $2, $0, 5
syscall

Sll $6, $5, 2
la $7, B
add $6, $7, $6
sw $2, ($6)

addi $5, $5, 1
j label_1
label_2:
jr $31
```

CLANG → Assembly MIPS

```
void readArrays(void){
   R5 = 0;
   label_1:
   R11 = (R5<16);
   if(R11 == 0) goto label_2;
      scanf("%d", &R2);
      A[R5] = (int)R2;
      scanf("%d", &R2);
      B[R5] = (int)R2;
      R5 = R5 + 1;
   goto label_1;
   label_2:
}
```

```
readArrays:
add $5, $0, $0
label_1:
slti $11, $5, 16
beq $11, $0, label_2

addi $2, $0, 5
syscall

sll $6, $5, 2
la $7, A
add $6, $7, $6
sw $2, ($6)

addi $2, $0, 5
syscall

sll $6, $5, 2
la $7, B
add $6, $7, $6
sw $2, ($6)

addi $5, $5, 1
j label_1
label_2:
jr $31
```

CLANG → Assembly MIPS

```
void printArray(void){

    R5 = 0;
    label_3:
    if(!(R5<16)) goto label_4;
        R6 = (long long)C[R5];
        printf("%d", (int)R6);
        printf(" ");
    R5 = R5 + 1;
    goto label_3;
    label_4:

}
```

```
printArray:
add $5, $0, $0
label_3:
slti $11, $5, 16
beq $11, $0, label_4

sll $6, $5, 2
la $7, C
add $6, $7, $6
lw $4, ($6)
addi $2, $0, 1
syscall

la $4, s
addi $2, $0, 4
syscall

addi $5, $5, 1
j label_3
label_4:
jr $31
```

CLANG → Assembly MIPS

```
void addArrays(void){
   R5 = 0;
   label_5:
   if(!(R5<4)) goto label_6;
      R6 = 0;
      label_7:
      if(!(R6<4)) goto label_8;
         /*internal block,
           following slide */
      R6 = R6 + 1;
      goto label_7;
      label_8:
   R5 = R5 + 1;
   goto label_5;
   label_6:
}
```

```
addArrays:
add $5, $0, $0
label_5:
slti $11, $5, 4
beq $11, $0, label_6
add $6, $0, $0
label_7:
slti $11, $6, 4
beq $11, $0, label_8
#internal block
#following slide
addi $6, $6, 1
j label_7
label_8:
addi $5, $5, 1
j label_5
label_6:
jr $31
```

CLANG → Assembly MIPS

```
R7 = (R5<<2);
R7 = R7 + R6;
R7 = A[R7];
R8 = (R6<<2);
R8 = R8 + R5;
R8 = B[R8];
R7 = R7 + R8;
R8 = (R5<<2);
R8 = R8 + R6;
C[R8] = R7;
```

```
sll $7, $5, 2
add $7, $7, $6

la $9, A
sll $10, $7, 2
add $9, $10, $9
lw $7, ($9)

sll $8, $6, 2
add $8, $8, $5

la $9, B
sll $10, $8, 2
add $9, $10, $9
lw $8, ($9)

add $7, $7, $8
sll $8, $5, 2
add $8, $8, $6

la $9, C
sll $10, $8, 2
add $8, $9, $10
sw $7, ($8)
```

MIPS Calling Conventions

- $zero
- $at, assembler temporary
- $v0-1, value registers, caller saved
- $a0-3, function arguments, caller saved
- $t0-9, temporary, caller saved
- $s0-7, saved registers, callee saved
- $k0-1, kernel registers
- $gp, global pointer, callee saved
- $sp, stack pointer, callee saved
- $fp, frame pointer, callee saved
- $ra, return address

CLANG Συμβάσεις

- Σε αυτό το εργαστήριο, στη CLANG δεν θα έχουμε global μεταβλητές R0-R31.
- οι GLOBAL μεταβλητές που θα έχουμε είναι:
  - μεταβλητή ZERO ίση με μηδέν.
  - μεταβλητές V0-V1 που περνάμε τις τιμές επιστροφής των συναρτήσεων
  - στις μεταβλητές A0-A3 περνάμε τις παραμέτρους των συναρτήσεων
  - στις μεταβλητές T0-T9 και S0-S7 τοποθετούμε όλες τις υπόλοιπες τιμές
  - τη μεταβλητή RA η οποία δεν θα έχει κάποιο πρακτικό σκοπό, αλλά θα πρέπει να τη διαχειρίζεστε σε CLANG όπως τον $ra σε assembly.

CLANG Συμβάσεις

- Στη CLANG οι παράμετροι των συναρτήσεων και οι τιμές επιστροφής θα πρέπει να περνάνε στις αντίστοιχες μεταβλητές
  - void function1(void)
  - εξαιρέσεις αποτελούν οι scanf, printf

CLANG Στοίβα

- Στη CLANG η στοίβα μπορεί να χρησιμοποιηθεί μέσω τοπικών μεταβλητών
- Στην αρχή της συνάρτησης ορίζουμε ένα πίνακα με θέσεις όσες και οι τετράδες που θα δεσμεύαμε στη στοίβα του MIPS
- Εκεί βάσει των συμβάσεων για τους Temporary και Saved Registers τοποθετούμε τις τιμές των T0-T9 και S0-S7 αντίστοιχα
- Στη CLANG είναι υποχρεωτικό να αποθηκεύετε την τιμή του RA στη στοίβα σε κάθε κλήση συνάρτησης
  - για πιο εύκολη μετάβαση σε assembly MIPS

Caller/Callee Registers

- Μπορείτε να χρησιμοποιήσετε είτε saved είτε temporary καταχωρητές μέσα στο σώμα της κάθε συνάρτησης
- Κάθε συνάρτηση που καλείται αδιαφορεί για το προηγούμενο περιεχόμενο των temporary καταχωρητών
  - Αποθήκευση πρίν την κλήση της συνάρτησης
  - Επαναφορά μετά την κλήση της συνάρτησης
- Κάθε συνάρτηση που καλείται οφείλει να κρατήσει το προγούμενο περιεχόμενο των saved καταχωρητών
  - Αποθήκευση στη στοίβα πρίν τη χρήση
  - Επαναφορά μετά τη χρήση

Παράδειγμα CLANG

```
void foo(void){
    long long stack[4];//we allocate stack space of 16 bytes
    stack[0] = S0; //save $s0
    stack[1] = S1; //save $s1
    /*The code of the function foo body is placed here*/
    stack[2]= T0;
    stack[3]= RA;
    A0 = 5;          //foo2 parameter
    foo2();
    RA = stack[3]; //restore $ra
    T0 = stack[2]; //restore $t0
    /*The code of the function foo body is placed here*/
    S1=stack[1];   // restore $s1
    S0=stack[0];   //restore $s0
    //IN ASSEMBLY WE NEED TO RELEASE STACK SPACE IN THIS LINE
    V0 = T0;         //Return value
}
```

Παράδειγμα Assembly

```
foo:
addi $sp, $sp, -16 # we allocate stack space of 16 bytes
sw $s0, 0($sp)     # save $s0 before use
sw $s1, 4($sp)     # save $s1
    # The code of the function foo body is placed here
sw $t0, 8($sp)     # save $t0 before call
sw $ra, 12($sp)
li $a0, 5
jal foo2
lw $ra, 12($sp)    # restore $ra
lw $t0, 8($sp)     # restore $t0 after call
    # The code of the function foo body is placed here
lw $s1, 4($sp)     # restore $s1
lw $s0, 0($sp)     # restore $s0
addi $sp, $sp, 16  # we release stack space of 16 bytes
move $v0, $t0      # set return value
jr $ra
```