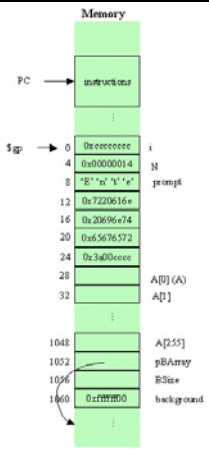
Today:

- MIPS Assembly Language Examples:
  - The Data/Memory Layout
  - Simple Expression
  - Array Expression
  - Inter-Statement Optimization
- A little bit about cebollita (if we have time)

Conventions for using register

- Register 0, always 0 (hardwired)
- PC starting from the first instruction
- Register $t0 - $t9 for your temporary use.
- $gp points to a memory space for program variables.



```
// none of these allocate any storage
#define MAX_SIZE 256
#define IF(a) if (a) {
#define ENDIF }
typedef struct {
    unsigned char red; // 'unsigned char' is an unsigned, 8-bit int
    unsigned char green;
    unsigned char blue;
    unsigned char alpha;
} RGBa;

// these allocate storage
int i;
int N = 20;
char prompt[] = "Enter an integer:";
int A[MAX_SIZE];
int* pBArray;
int BSize;
RGBa background = {0xff, 0xff, 0xff, 0x0};
```

A Simple Expression

- C code:

```
i = N*N + 3*N
```

- Assembly code:

```
lw    $t0,  4($gp)           # fetch N
mult  $t0,  $t0,    $t0      # N*N
lw    $t1,  4($gp)           # fetch N
ori   $t2,  $zero,  3        # 3
mult  $t1,  $t1,    $t2      # 3*N
add   $t2,  $t0,    $t1      # N*N + 3*N
sw    $t2,  0($gp)           # i = ...
```

Optimization

- C code:

```
i = N*N + 3*N          (= N * (N + 3))
```

- Assembly code:

```
lw    $t0,  4($gp)           # fetch N
add   $t1,  $t0,    $zero    # copy N to $t1
addi  $t1,  $t1,    3        # N+3
mult  $t1,  $t1,    $t0      # N*(N+3)
sw    $t1,  0($gp)           # i = ...
```

Array Expression

```
# A[i] = A[i/2] + 1;
lw    $t0,  0($gp)              # fetch i
srl   $t0,  $t0,    1           # i/2
addi  $t1,  $gp,    28          # &A[0]
sll   $t0,  $t0,    2           # turn i/2 into a byte offset (*4)
add   $t1,  $t1,    $t0         # &A[i/2]
lw    $t1,  0($t1)              # fetch A[i/2]
addi  $t1,  $t1,    1           # A[i/2] + 1
lw    $t0,  0($gp)              # fetch i
sll   $t0,  $t0,    2           # turn i into a byte offset
addi  $t2,  $gp,    28          # &A[0]
add   $t2,  $t2,    $t0         # &A[i]
sw    $t1,  0($t2)              # A[i] = ...
# A[i+1] = -1;
lw    $t0,  0($gp)              # fetch i
addi  $t0,  $t0,    1           # i+1
sll   $t0,  $t0,    2           # turn i+1 into a byte offset
addi  $t1,  $gp,    28          # &A[0]
add   $t1,  $t1,    $t0         # &A[i+1]
addi  $t2,  $zero,  -1          # -1
sw    $t2,  0($t1)              # A[i+1] = -1
```

Inter-statement optimization

```
# A[i] = A[i/2] + 1;
lw      $t0,    0($gp)          # fetch i
srl     $t1,    $t0,    1       # i/2
sll     $t1,    $t1,    2       # turn i/2 into a byte offset (*4)
add     $t1,    $gp,    $t1     # &A[i/2] - 28
lw      $t1,    28($t1)         # fetch A[i/2]
addi    $t1,    $t1,    1       # A[i/2] + 1
sll     $t2,    $t0,    2       # turn i into a byte offset
add     $t2,    $t2,    $gp     # &A[i] - 28
sw      $t1,    28($t2)         # A[i] = ...
# A[i+1] = -1;
addi    $t1,    $zero, -1       # -1
sw      $t1,    32($t2)         # A[i+1] = -1
```

The assembly code generated by cebollita

```
#   a[i] = a[i/2] + 1
lw $t0, 8($fp)      #load i
addi $at, $0, 2
sllv $t0, $t0, $at  # byte offset
addi $t1, $fp, 36 # &A[0]
add $t1, $t1, $t0 # &A[i]
lw $t0, 8($fp)      # load i
addi $t2, $0, 2     # $t2=2
div $t3, $t0, $t2 # i/2
addi $at, $0, 2
sllv $t3, $t3, $at  # byte offset
addi $t0, $fp, 36   #&A[0]
add $t0, $t0, $t3 #&A[i/2]
lw $t0, 0($t0)      #load A[i/2]
addi $t2, $0, 1     # 1
add $t3, $t0, $t2 # A[i/2]+1
sw $t3, 0($t1)      #store
```

```
#   a[(i + 1)] = -1
lw $t0, 8($fp)      #load i
addi $t1, $0, 1
add $t2, $t0, $t1  #i+1
addi $at, $0, 2
sllv $t2, $t2, $at   #byte offset
addi $t0, $fp, 36  #&A[0]
add $t0, $t0, $t2  #&A[i+1]
addi $t1, $0, 0     #0
addi $t2, $0, 1     #1
sub $t3, $t1, $t2  # -1
sw $t3, 0($t0)      #store
```

Cebollita

- A toolkit that helps developing program on a MIPS-like ISA.
  - a C-like language (C--)
  - a compiler
  - an assembler
  - a linker
  - a loader
  - a software simulator

Cebollita con't

- Start/Program Files/Desktop Tools/cse378/ceb
- cebcc to compile C-- programs, (.c) to (.s) files
- cebasm to assemble assembly (.s) files into (.o) files,
- ceblink to link .o's into a.out's
- cebsim to run the simulator