|  |  |  |  |
| --- | --- | --- | --- |
| Unsigned: 0b0100=4  Signed: 0b1011=-5  Unsigned: 0x0382=898  Signed: 0xffff=-1 | .text - program text (your instructions)  .data - global variables (local variables go on the stack)  .bss - uninitialized data | Long int=8 bytes  Short int = 2 bytes  All pointers = 8 bytes | W registers only reference the last 32 bytes of a register, but can still be called with the corresponding X registers |
| **Logical Shift Left:** Every bit in the number shifts by a certain number of places and is padded with 0s on the end (same as multiplying by 2^n where n is the number of places)  00001001 << 2 = 00100100 (9 \* 2^2 = 36)  **Logical Shift Right:** The opposite of logical shift left where you pad with 0s (same as doing integer division by 2^n where n is number of places shifted)  00001001 >> 2 = 00000010 (9 / 2^2 = 2)  **Arithmetic Shift Right**: Similar to logical shift right except you pad with the MSB (most significant bit) so that the sign is preserved.  00001001 >> 2 = 00000010  11100000 >> 2 = 11111000 (-32 / 2^2 = -8)  When shifting with hexadecimal, if its aratmetic, if the leading digit is 0-7, pad with 0, 8-f, pad it with f | int a = 12; // 1100  int b = 6;  // 0110  Bitwise AND: &  printf("%d\n", a & b)  // 0100  Bitwise OR: |  printf("%d\n", a | b)  // 1110  LEFT SHIFT: <<  printf("%d\n", a << 1) // 11000  RIGHT SHIFT: >>  printf("%d\n", a >> 1) // 0110  char\* arr = "382 Midterm!";  // What do each of these print?  putchar(\*arr); = “3”  putchar(\*(arr + 1));=”8”  putchar(\*(arr + 2));=”2”  putchar(\*(arr + 11));=”!” | Caller:  Put parameters in X0 – X7, and more on stack if necessary;  Save Caller-saved registers;  Retrieve return value from X0;  Callee:  Sets the Stack pointer:  Subtract SP to make space on stack;  Save callee-saved registers;  Save X30 if non-leaf procedure;  Put return value in X0;  (and subsequent registers up to X7 if applicable, then memory – usually not applicable)  Removes the stack pointer:  Restore X30 and callee-saved registers;  Add SP to deallocate the frame;  RET | C code:  long cat(long x, long y)  {  if (x < y) return dog();  else return bunny(); }  Assembly code:  Cat:  CMP X2, X3  B.LT Bod  BL Bunny  Ret  Dog:  BL exit  RET  Bunny:  BL exit  RET  Start:  ADR X0, x  ADR X1, y  LDR X2, [X0]  LDR X3, [X1]  BL Cat  RET  exit:  MOV X0, 0  MOV X8, 93  SVC 0 |
| ARM-V8 assembly provides 32 general purpose 64-bit integer registers, each of which can store a single value. They are labelled X0-X31. | The lower (least significant) 32 bits of these registers can be accessed with a W alias. For example, W0 refers to the lower 32 bits of register X0. | There are also specialized registers such as program counter PC, which cannot be modified in any special way by most instructions (PC is normally just incremented by 4 every instruction, except for branch instructions). |  |
| X0 – X7: Parameters  X8 – X18: Temporary registers (Caller-saved)  X19 – X29, X30: Callee-saved registers  X0, return value register  X30, LR (link register)  X31, XZR (zero register) | unsigned/signed – char,int  short – int  long – int,double  char\* : null terminated char pointer  putchar(char)          : write raw character  puts(string)           : write raw string  printf(format, args…)  : write formatted string |  |  |

A black background with pink letters

Description automatically generatedA black background with pink text

Description automatically generatedA white rectangular box with black text

Description automatically generatedA table with different types of words

Description automatically generatedA diagram of a stack

Description automatically generatedA table with numbers and instructions

Description automatically generated

A black and white rectangular sign with numbers

Description automatically generatedA black and white rectangular sign with numbers

Description automatically generated

A screenshot of a computer

Description automatically generated