Abstract Robot Control Language (ARCL)

This is the intermediate language used for SCARL. This language is structed as to be suitable to be converted to either assembler or to code that is used for robotics control. The abstract machine is one that utilizes ports and has a simple memory layout. We define the instruction set for the Abstract Robot here and provide the instruction’s close relative within the Atmel AVR Instruction Set.

Important in this language is the flags for the SCARL linker that says to generate code for a specific device action. It is up to the SCARL linker (it will be a compiler-linker combination) to perform the final conversion when it runs into such a directive, as the implementation of the device may vary from machine to machine.

Description of the abstract machine that ARCL runs on:

A machine that:

1. Has a 8 general purpose 16 bit registers.
2. Has a special implicit frame pointer register
3. Special register for returning function values called “FRR”
4. Has a stack with unbounded memory
5. Has unbounded heap space.
6. Has a space of read-only memory that is set before execution of the program. This is where program code is loaded into.
7. Each atomic object is a 2 byte word that can be interpreted in multiple different ways
8. Has a set of peripheral pins that may be set to input or output, and have power on, or read the power from it

Instruction Set Table

|  |  |
| --- | --- |
| **Expression Instructions** | **Action** |
| ADD *reg1 reg2* | Add the two registers, storing the result in the right hand side register |
| SUB *reg1 reg2* | Subtract the two registers *reg1 – reg2*, storing the result in the right hand side register |
| MULT *reg1 reg2* | Multiply the two registers, storing the result in the right hand side register |
| DIV *reg1 reg2* | Integer division on the two registers *reg1 / reg2*, storing the result in the right hand side register |
| FLIP *reg1* | Flip the sign of the value in the given register. |

|  |  |
| --- | --- |
| **Memory Instructions** | **Action** |
| MOV *src dst* | Moves data from the *src* register to the *dst* register |
| LOADL *reg lit* | Loads the register *reg* with the literal *lit* |
| LOADF *reg offset* | Loads the value stored *offset* values away from the top of the stack frame and loads it into register *reg* |
| LOADR *reg offset* | Finds the value from storage that is *offset* values away from the beginning of static storage and stores it into register *reg* |
| STORF *reg offset* | Goes to the value located *offset* values away from the top of the stack frame and changes it so that it is equal to the value located in the register *reg* |
| STORR *reg offset* | Loads the value in register *reg* into the spot that is *offset* values away from the beginning of storage. |
| FRAMEU *offset* | Push the stack frame so that it is *offset* values farther away. For setting up activation record space |
| FRAMEO *offset* | Pop the stack frame so that it is *offset* values closer. For removing activation record space |
| RSAVE | Save the current state of the machine by stacking all registers on the stack and incrementing the frame pointer. |
| RLOAD | Recovering a previous state by unstacking previously saved registers and then decrementing the frame pointer appropriately |

|  |  |
| --- | --- |
| **Program Control Instructions** | **Action** |
| LABEL *label* | Defines a label at this spot and calls it *label* |
| PROC *label* | Begins a procedure definition and calls it *label* |
| RET | Returns control back to the callee from a procedure call |
| CALL *label* | Stacks the current instruction address as a return point and then transfers control to *label* |
| JMP *label* | Jumps to the label *label* |
| CMP *reg1 reg2* | Compares the two registers *reg1* and *reg2* and changes the internal state of this ARCL machine. (see below) |
| JPE *label* | Jump to the *label* if the most recent CMP had *reg1 == reg2* |
| JPNE *label* | Jump to *label* if the most recent CMP had *reg1* != *reg2* |
| JPGR *label* | Jump to the *label* if the most recent CMP had *reg1* > *reg2* |
| JPLS *label* | Jump to the *label* if the most recent CMP had *reg1 < reg2* |
| JPGE *label* | Jump to the *label* if the most recent CMP had *reg1* >= *reg2* |
| JPLE *label* | Jump to the *label* if the most recent CMP had *reg1 <= reg2* |
| JAND *label* | Jump to *label* if the most recent CMP evaluates to true under the following conditions:   |  |  |  | | --- | --- | --- | | *reg1* | *reg2* | evaluation | | not 0 | not 0 | true | | 0 | not 0 | false | | not 0 | 0 | false | | 0 | 0 | false | |
| JOR *label* | Jump to *label* if the most recent CMP evaluates to true under the following conditions:   |  |  |  | | --- | --- | --- | | *reg1* | *reg2* | evaluation | | not 0 | not 0 | true | | 0 | not 0 | true | | not 0 | 0 | true | | 0 | 0 | false | |
| DELAY *reg* | Delays the program for the number of milliseconds as specified in *reg* |

General purpose status register? Look into this. (Zero flag, carry flag, etc) and this would make it easier to implement it on a variety of actual machines.

|  |  |
| --- | --- |
| **Device Control Instructions** | **Action** |
| OUTPUT *reg* | Sets the pin as specified by the register *reg* to be an output pin. |
| INPUT *reg* | Sets the pin as specified by the register *reg* to be an input pin. |
| WRITE *regPin regData* | Writes the value specified in *regData* to the pin specified by *regPin*. Only valid if the pin is specified as an output pin. Generally, 0 is off and 1 or higher is on. Different values in *regData* may specify degrees of power, depending on the target architecture. |
| READ *regPin regData* | Reads the data recognized by the pin specified by *regPin* and places it into the *regData* register. Only valid if the pin is an input register. Behavior depends on if the pin is digital or analog, which depends on the pin selected by the user. |

**Required Files for Linking**

When SCARL code is compiled, there may be non-executable lines of the form @@@290?identifier@@@, which is a signal to the linker to define functions for the identifier based on the type indicated before the ?.

**The ARCL Machine In-Depth**

How an ARCL compliant machine is implemented does not matter as long as it can accept the instructions and perform actions as given above. The only way it can run ARCL code directly is through software since the encoding of the instructions is dependent on the character set used. When an instance of the machine is running, it may be given an ARCL file, which it will read and internally decipher into its own encoding before executing, but since the implementation details of that are encapsulated to the ARCL machine itself, it is still an ARCL compliant “machine”.

The list of register names that are valid are:

1. R0
2. R1
3. R2
4. R3
5. R4
6. R5
7. R6
8. R7
9. FRR (function return register)

Internally there may also be a frame pointer, but that may not be referred to explicitly.The memory given for both the stack and the heap may be implementation dependent. The implementation should not add additional registers. Memory is to be organized through two-byte words, and each register is to have a word of space. Input and output peripherals are known as device pins, and each must have a number from 1 up to the number of actual pins. The input or output mode of a pin is dependent on the function it is assigned by the linker with placing INPUT and OUTPUT instructions in the place where a device is originally declared.

We now see that it is necessary to create a document describing the implementation and usage of the scarlrvm program.