A VLSI Implementation of a Stack-Frame Computer

C. Longway, Ray Siferd, and R. D. Dixon
Wright State University

Introduction

The implementation of a 32-bit computer, the SF1 (the architecture is described in the companion paper by Dixon), is described in this paper. A diagram of the implementation, which is pipelined to a depth two, is at the end of this paper, and all names used here come from that diagram or from Dixon's paper.

This work is being done by graduate students and has been separated into distinct functions for each chip. Later versions will require fewer chips. We expect a 125 nanosecond version to be available by spring 1988, and our goal for 1989 is a 20M-instruction/s machine.

ALU CHIP IMPLEMENTATION

The ALU is implemented in a VLSI chip and is the site of all operations on data. One operation will take place during each instruction cycle. At the completion of an instruction cycle, the output of the ALU register is stored in the TOS. During the first portion of the next instruction cycle, the output stored in the TOS is transferred to the OTOS to be use as an operand to the ALU register or output to the SBUS. During the last portion of an instruction cycle, the ALU chip places a value on the SBUS. This value can be selected from the OTOS, which stores values from previous operations, or the ALUI register, which holds the SBUS value used as input for the current operation.

For binary operations, the OTOS register is used as one operand. The ALUI register is used as the other operand which is loaded from the SBUS during the first portion of an instruction cycle. Only one operand can be accessed from a stack during a instruction cycle. However, binary operations can be completed during one cycle since one operand is supplied internally by the OTOS register.

STACK-FRAME CHIP IMPLEMENTATION

The logic on the SBUS includes stack-frame chips as well as special purpose chips and hardware. On every main memory cycle, the SBUS goes through a read cycle and a write cycle.

The stack-frame VLSI chips can be thought of as memory as well as stacks. The stack-frame chips support the typical stack functions of PUSH and POP as well as the typical memory functions of READ and WRITE. During the first portion of an instruction, either a READ or a POP can be performed. During the last portion of an instruction, a WRITE or a PUSH can be performed.

No addresses or pointers are used for the PUSH or POP operations because they are implemented in hardware as vertical shift registers. For the PUSH or POP instruction, the entire contents inside the stack memory chip will move one address location. However, the memory functions of READ and WRITE require that an explicit address be supplied.

The special purpose stack-frames and hardware are accessed in the same manner as the stack-frames. The C frame functions as a special purpose register to pass constant values from the the address field of an instruction to the SBUS. The I/O frame be used for I/O, but also can potentially be used for ALU operations that do not fit in the ALU

chip. The PC (program counter) is also treated as a stack by the stack control signals. The implementation of this VLSI chip is discussed below.

PC CHIP IMPLEMENTATION

The PC is implemented through the following functions: RESET, INCREMENT, JUMP, JSR (JUMP SUBROUTINE), RETURN, and EXTEND.

RESET Function. During a system reset, the INC (incremented) output is forced to zero and allowed to propagate to the input of the INC and out the PCO (program counter output) driver to address memory location zero.

INCREMENT Function. When the reset signal is released, the INC will output an INC value to the PCO for addressing consecutive words in memory.

JUMP/JSR Function. When an instruction is placed on the MBUS with the least significant bit being zero, the PC passes this instruction through JUMP and immediately wraps this value around to PCO to be used as the next address. At the same time, the PC is loading INC with the new address. If, in addition to bit zero, the JSR bit it is set to zero, the PC responds by placing the output of INC on the stack bus to be stored as the return address.

RETURN Function. During implementation of a RETURN function, the return address is accessed from any device on the SBUS. The return address is sent to the ALUI register during the first portion of the instruction cycle. During the second portion of the cycle, the return address is again placed on the SBUS and then propagated through POP to PCO. At the same time that RETURN is being propagated to PCO, it is also being loaded into the input of the INC.

EXTEND Function. The EXTEND function can be activated during the first portion of any instruction cycle. When activated, the PCO driver is disabled to allow the ALU chip to control the MBUS.

During a JUMP/JSR or RETURN it is necessary to move the input values to the PCO very quickly. The internal architecture has been optimized to implement these instructions quickly. Simulations show that the data propagation of these instructions will be less than 5ns. The majority of the PC circuitry is used to perform the 32-bit increment. This INC is composed of combinational logic and will stabilize in less then 10 ns.

CONTROL UNIT IMPLEMENTATION

In the current phase of design, control is implemented in PAL's. We have selected to use PAL's since they allow us to make changes in control at a low cost and fast turnaround time compared to having the control logic in VLSI. The PAL's offer 10nS setup and delay times which allow the system to run at full speed. During later phases of design, we expect the control to migrate into the VLSI with the option of using either internally generated or externally generated control.

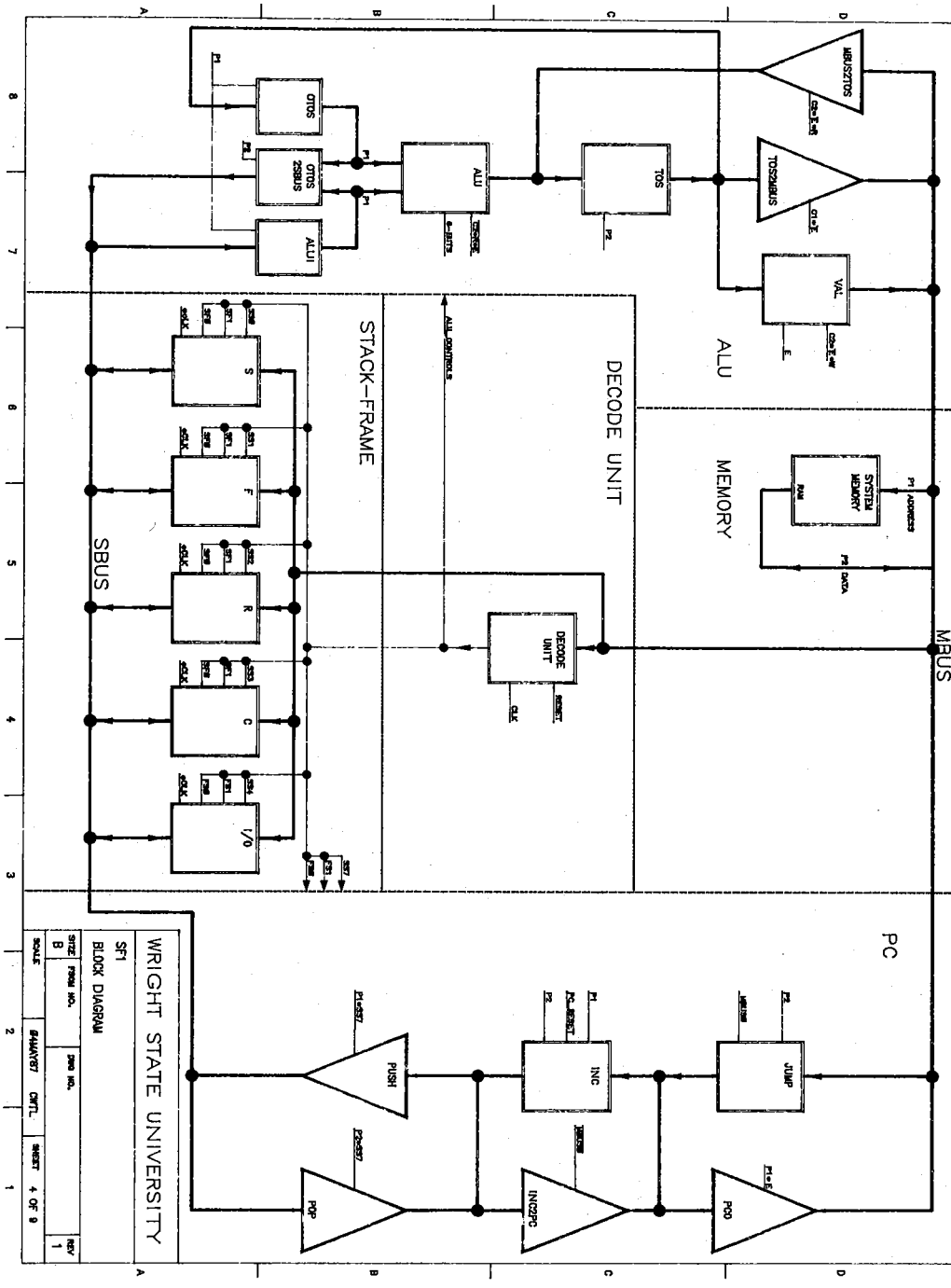
The extended instructions require a minimal amount of state to sequence through the extended portion of the instruction, but otherwise each cycle is controlled directly by the bits from the current instruction.

A Coding for the SF1

The description of the instruction set of the SF1 in Dixon was made without reference to the mapping between instructions and the binary coding of those instructions. The mapping given here is being used in the current implementation and is given as an example.

An instruction is composed of the following fields,
<JSR or operation bit><the rest>,
which are mapped to bits B31 through B0 (B31 is the high order bit).
<JSR or operation bit> : B0
<the rest> : B31-B1
In case B0=0, the instruction is a JSR or a JMP and the fields are defined as
<the rest> := <word number><JMP bit>
<word number> : B31-B2
<JMP bit> : B1
The instruction is a JSR if B1=0, a JMP if B1=1. In either case the address to which control is transfered is
<word number>*4.
In the case B0=1, the instruction is a regular operation with fields:
<the rest> := <opcode><source><destination>
<displacement><direct><memory>
<opcode> := <op><status>
<op> : B7-B3
<status> : B2
Here <op> is a binary coding for ADD, SUB, SUBR, AND, OR, XOR, NOOP, LOAD, SHIFTRL, SHIFTRA, SHIFTL.
If <status> is 0 then a regular operation is indicated. If <status> = 1 then the corresponding _ST instruction is indicated (returns the status of the operation in TOS).

<source> := <s-stack-frame><s-mode>
<s-stack-frame> : B15-B13
<s-mode> : B12
<destination> := <d-stack-frame><d-mode>
<d-stack-frame> : B11-B9
<d-mode> : B8
The <?-stack-frame> fields are a 3-bit coding of the stack-frame names, S, F, R, L, G, C, I, P.
The <?-mode> is 0 for stack access and 1 for frame access.
<displacement> : B28-B16
This field is used as the address into any stack-frame accessed in frame mode.
<direct> : B1
The <direct> field is a 0 when the old TOS is used to provide the value to be written in the destination. If it is a 1 then the ALUI, the value just read from the source, is used.
<memory> := <extended><read/write>
<extended> : B31
<read/write> : B30
When <extended> is a 1 then this is an extended instruction which occupies two cycles and does a main memory access. The <read/write> bit is a 0 for a read, a 1 for a write.


MBUS
ALU
MEMORY
DECODE UNIT
STACK-FRAME
SBUS
PC
TOS
ALU
OTOS
ALU1
VAL
SYSTEM MEMORY
DECODE UNIT
S
F
R
C
I/O
PUSH
INC
JUMP
POP
INC2PC
POD
WRIGHT STATE UNIVERSITY
SF1
BLOCK DIAGRAM